Decomposition of Index Generation Functions Using a Monte Carlo Method

Tsutomu Sasao
Dept. of Computer Science
Meiji University
Kawasaki 214-8571, Japan

Jon T. Butler
Dept. Electrical and Computer Engr.
Naval Postgraduate School
Monterey, CA, 93943-5121, U.S.A

March 8, 2017

Abstract

This chapter considers functional decompositions of index generation functions. A Monte Carlo method to predict column multiplicities of the decomposition charts is presented. With this, we can efficiently find a circuit structure to implement the function. Our goal is not to find a minimum column multiplicity of a given function, but to predict the column multiplicities of random index generation functions. With these results, we can estimate the size of the programmable architecture when we know only the number of inputs and registered vectors.

Keywords: Functional decomposition, Statistical method, Index generation function, Stirling number, Reconfigurable logic, Monte Carlo method

1 Introduction

One of the important tasks in logic synthesis is to find a circuit structure that is suitable for implementation. **Functional decomposition** [1, 4] is a technique to decompose a circuit into two subcircuits each with a lower cost than the original circuit. Various techniques to find functional decompositions have been presented [2, 6, 8, 10]. They are routinely used in logic synthesis. Many circuits that are used in computers have some structure, and they are not random [3]. On the other hand, almost all randomly generated switching functions have no effective decompositions.

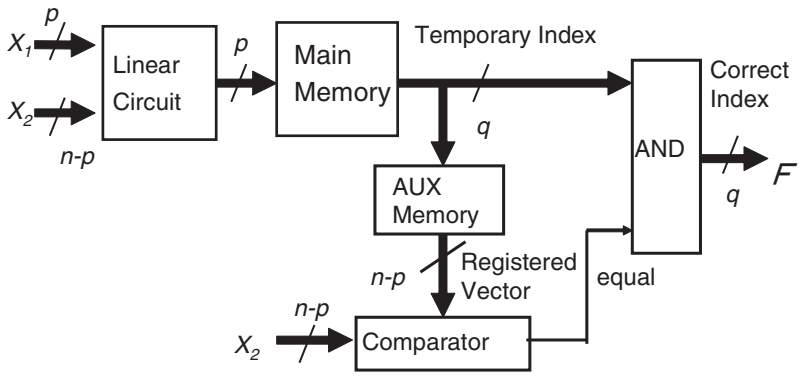

Figure 1.1: Index generation unit (IGU).

Recently, we are working on index generation functions [13, 14]: $f : \{0,1\}^n \rightarrow \{0,1,2,\ldots,k\}$, where $k \ll 2^n$. Here, n is the number of bits in each registered vector, and k is the total number of registered vectors.

They are used for access control lists, routers, and virus scanning for the internet, etc.. Index generation functions found in practical applications have properties similar to those of randomly generated index generation functions.

In this chapter, we show that index generation functions often have effective decompositions. To show this, we use a Monte Carlo method to predict the column multiplicity of the decomposition charts for random index generation functions.

An index generation function can be efficiently implemented by an LUT or an **IGU** (Index Generation Unit, Fig. 1.1), which are programmable [13]. We omit the explanation of the operation of an IGU, which appeared in [14] [1]. Currently, an IGU is implemented by a combination of field programmable gate arrays (FPGAs) and memories [9]. However, a single-chip custom LSI can also be used to implement an index generation function. Suppose that LSIs for IGUs with n inputs and weight k are available. For a function with larger k, we can partition the set of vectors into several sets, and implement each by an independent IGU. The outputs of the IGUs can be combined by an OR gate to produce the final output [15]. This is a **parallel decomposition** (Fig. 1.2). On the other hand, for a function with larger n, we can partition the set of input variables into two sets X_1 and X_2 to produce the function. This is a **serial decomposition** (Fig. 1.3).

In our applications, when we prepare the programmable IGU chip, we only know the values of n and k, but do not know the detail of the functions. Functions to be imple-

[1] For simplicity, readers can assume that LUTs are used to implement the functions.

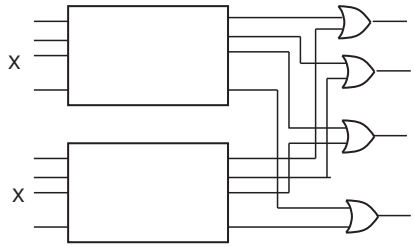

Figure 1.2: Realization of a function by a parallel decomposition.

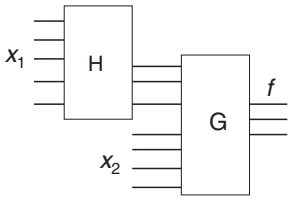

Figure 1.3: Realization of a logic function by serial decomposition.

mented are different for different users. This situation is similar to the case of FPGAs: FPGA companies do not know all the functions to implement in advance. In this chapter, we try to predict the complexity of random functions.

The rest of the chapter is organized as follows: Section 2 introduces functional decomposition. Section 3 introduces the properties of index generation functions. Section 4 shows a Monte Carlo method to derive column multiplicity of decomposition charts for random index generation functions. Section 5 shows a procedure for computing the column multiplicity of decompositions. Section 6 shows the experimental results. Section 7 shows a method to assess the programmable architecture for index generation functions. Section 8 concludes the chapter.

2 Decomposition

In this part, we introduce basic concepts of functional decomposition.

Definition 2.1 *[1] Let $f(X)$ be a function, and (X_1, X_2) be a partition of the input variables, where $X_1 = (x_1, x_2, \ldots, x_s)$ and $X_2 = (x_{s+1}, x_{s+2}, \ldots, x_n)$. The* **decomposition chart** *for f is a two-dimensional matrix with 2^s columns and 2^{n-s} rows, where each column and row is labeled by a unique binary assignment of values to the variables. Each assignment maps under f to $\{0, 1, \ldots, k\}$. The function represented by a column is a*

		0	0	1	1	x_1
		0	1	0	1	x_2
0	0	0	0	0	1	
0	1	1	1	0	0	
1	0	0	1	0	0	
1	1	0	0	0	0	
x_3	x_4					

Figure 2.1: Decomposition chart of an logic function.

column function *and is dependent on* X_2. *Variables in* X_1 *are* **bound variables**, *while variables in* X_2 *are* **free variables**. *In the decomposition chart, the* **column multiplicity**, *denoted by* μ, *is the number of different column functions.*

Example 2.1 *Fig. 2.1 shows a decomposition chart of a* 4*-variable switching function.* $X_1 = (x_1, x_2)$ *denotes the bound variables, and* $X_2 = (x_3, x_4)$ *denotes the free variables. Since all the column patterns are different and there are four of them, the column multiplicity is* $\mu = 4$. ∎

Theorem 2.1 *[4] For a given function* f, *let* X_1 *be the bound variables, let* X_2 *be the free variables, and let* μ *be the column multiplicity of the decomposition chart. Then, the function* f *can be represented as* $f(X_1, X_2) = g(h(X_1), X_2)$, *and is realized with the network shown in Fig. 1.3. The number of signal lines connecting blocks* H *and* G *is* $r = \lceil \log_2 \mu \rceil$, *where* H *and* G *realize* h *and* g, *respectively.*

The logic functions for H and G can be realized by memories, and the complexities for G and H can be measured by the number of bits in the memories. The signal lines connecting H and G are called **rails**. When the number of rails r is smaller than the number of input variables in X_1, it is **support-reducing** [5], and we can often reduce the total amount of memory to realize the logic in Fig. 1.3.

3 Index Generation Functions and Their Properties

Definition 3.1 *Consider a set of* k *different binary vectors of* n *bits. These vectors are* **registered vectors**. *For each registered vector, assign a unique integer from* 1 *to* k, *called an* **index**. *A* **registered vector table** *shows, for each registered vector, its index. An*

Table 3.1: Registered vector table.

Vector				Index
x_1	x_2	x_3	x_4	
1	0	0	1	1
1	1	1	1	2
0	1	0	1	3
1	1	0	0	4

Table 3.2: Index generation function.

Input				Output		
x_1	x_2	x_3	x_4	y_1	y_2	y_3
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	**0**	**1**	**1**
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	**0**	**0**	**1**
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	**1**	**0**	**0**
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	**0**	**1**	**0**

index generation function *f produces the corresponding index if the input matches a registered vector, and produces 0 otherwise. k is the* **weight** *of the index generation function. An index generation function represents a mapping: $f : B^n \rightarrow \{0, 1, 2, \ldots, k\}$, where $B = \{0, 1\}$.*

Example 3.1 *Table 3.1 shows a registered vector table with $k = 4$ vectors. The corresponding index generation function is shown in Table 3.2. In this case, the output is represented by 3 bits. So, it shows a mapping $B^4 \rightarrow \{0, 1, 2, 3, 4\}$. Note that the index values from Table 3.1 are shown in binary in bold. Also, note that registered vectors missing in Table 3.1 are shown in Table 3.2 mapped to 000.* ■

Typically, k is much smaller than 2^n, the total number of input combinations.

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

Figure 3.1: Decomposition chart for f.

Example 3.2 *Consider the decomposition chart in Fig. 3.1. It shows an index generation function $f(X)$ with weight 7. $X_1 = (x_1, x_2, x_3, x_4)$ denotes the bound variables, and $X_2 = (x_5)$ denotes the free variable. Note that the column multiplicity of this decomposition chart is 7.* ■

Lemma 3.1 *Let $\mu(f(X_1, X_2))$ be the column multiplicity of a decomposition chart of an index generation function f, let k be the weight of f, and let s be the number of variables in X_1. Then,*

$$\mu(f(X_1, X_2)) \leq \min\{k + 1, 2^s\}.$$

(Proof) Since the number of non-zero outputs is k, the column multiplicity never exceeds $k + 1$. Further, the column multiplicity never exceeds the total number of columns, 2^s. □

Lemma 3.2 *Let f be an index generation function with weight k. Then, there exists a functional decomposition $f(X_1, X_2) = g(h(X_1), X_2)$, where g and h are index generation functions, such that the weight of g is k, and the weight of h is at most k.*

(Proof) Consider a decomposition chart, in which X_1 denotes the bound variables, and X_2 denotes the free variables. Let $X_1 = (x_1, x_2, \ldots, x_s)$, where $s \geq \lceil \log_2(k + 1) \rceil$. Let h be a function where the variables are X_1, and the output values are defined as follows: Consider the decomposition chart, where assignments of values to X_1 label columns (i.e., bound variables). For the assignments to X_1 corresponding to columns with only zero elements, $h = 0$. For other assignments, the outputs of h are distinct integers from 1 to w_h, where w_h denotes the number of columns that have non-zero element(s). Since $w_h \leq k$, the weight of h is at most k, and the number of output values of h is at most $k+1$. On the other hand, the function g is obtained from f by reducing some columns that have all zero output in the decomposition chart. Thus, the number of non-zero outputs in g is equal to the number of non-zero outputs in f. Thus, g is also an index generation function with weight k. □

Table 3.3: Truth table for h.

x_1	x_2	x_3	x_4	y_1	y_2	y_3
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	0	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	1	1	0
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

	0	0	0	0	1	1	1	1	y_1
	0	0	1	1	0	0	1	1	y_2
	0	1	0	1	0	1	0	1	y_3
0	0	1	0	3	4	5	0	0	
1	0	0	2	0	0	6	7	0	
x_5									

Figure 3.2: Decomposition chart for g.

Example 3.3 *Consider the decomposition chart in Fig. 3.1. Let the function $f(X)$ be decomposed as $f(X_1, X_2) = g(h(X_1), X_2)$, where $X_1 = (x_1, x_2, x_3, x_4)$, and $X_2 = (x_5)$. Table 3.3 shows the function h. It is a 4-variable 3-output logic function with weight 6. The decomposition chart for the function g is shown in Fig. 3.2. As shown in this example, the functions obtained by decomposing the index generation function f are also index generation functions, and the weights of f and g are 6 and 7, respectively.* ■

4 Balls into Bins Model

In this part, we show a method to predict the column multiplicity of a functional decomposition using a balls into bins model.

In Definition 2.1, we specified that the first s variables $x_1, x_2, \ldots, x_s$ are in X_1, and the remaining $n - s$ variables are in X_2. However, in many cases, we can select any s variables for X_1. In this case, the problem of functional decomposition is to partition the variables into two sets, so that the number of rails r between two blocks is minimized. To do this, we want to reduce μ. Note that, in order to reduce the rails between H and G (see Fig. 3.1), we must reduce μ so that it is equal to or less than the smallest power of two.

Given a function table, we have to search $\binom{n}{s}$ combinations, where n is the total number of variables, and s is the number of the bound variables. From Lemma 3.1, we have an upper bound on μ:

$$\mu(f(X_1, X_2)) \leq \min\{2^s, k+1\}.$$

First, we show an exhaustive approach to find a decomposition.

Example 4.1 *Consider the registered vector table shown in Table 4.1. It is a* 20*-variable random index generation function with weight* $k = 20$*. We need to find an effective decomposition that reduces the implementation cost. The number of decompositions is equal to the number of ways to partition the set of variables into two non-empty sets, that is* $2^n - 2$*. If we know the expected column multiplicity, then we can predict how likely exhaustive search can find a good decomposition. Suppose that the number of bound variables is* $s = 8$*. In this case, the decomposition chart has* $2^8 = 256$ *columns and* $2^{20-8} = 2^{12} = 4096$ *rows. Then, the number of decompositions to check is* $\binom{20}{8} = 125970$*. By exhaustive search, we find the minimum column multiplicity to be* $\mu = 15 + 1 = 16$*. One decomposition was found with this column multiplicity; it has for the bound variables* $X_1 = (x_1, x_2, x_5, x_7, x_{10}, x_{13}, x_{14}, x_{17})$*. In this case, the number of rails is reduced to four (from five).* ∎

An approach in which we generate a random index generation function, cast it into a decomposition chart with one of many choices for the bound variables, compute the column multiplicity, and choose the minimum is computationally too expensive. To estimate the distribution of column multiplicities, we need a more efficient method.

The column multiplicity μ of an index generation function with weight k can be predicted by the **balls into bins model** as follows: In the decomposition chart of an index generation function, all care values occur in exactly one of the columns. Since all care values are distinct, any column with a care value is distinct from all other columns. The only columns that are identical are those that contain no care values. Therefore, the column multiplicity of a decomposition chart is just the number of columns with at least one care values plus 1 if there is at least one column with no care values.

Consider another random process in which there are as many bins as there are *columns* in the decomposition chart. We can distribute k distinct balls into bins with 0,1,..., balls

allows in each bin. The random distribution of balls in this way is equal to choosing a bin number with repetition, one for each ball. If repetition occurs, then multiple balls fall into the same bin. Assume that there are 2^s distinct bins, and k distinct balls are randomly thrown into these bins. If all the balls fall into the same bin[2], then $\mu = 1 + 1 = 2$. If all the balls fall into k different bins, then $\mu = \min\{k + 1, 2^s\}$. However, in most cases, $2 < \mu \leq k + 1$. The number of non-empty bins plus one is equal to the column multiplicity μ.

An efficient method to simulate the balls and bin model is the **integer set model** as follows: Assume that f k integers represented as standard binary numbers on s bits are randomly generated.

Example 4.2 *Let us predict the column multiplicity for random index generation functions of $n = 20$ variables and weight $k = 20$ by Monte Carlo simulation using the integer set model. The number of different n variable index generation functions with weight k is $P(2^n, k) = \frac{2^n!}{(2^n-k)!}$. For $n = 20$ and $k = 20$, this is about 2.6×10^{120}, which is too large to search exhaustively.*

We can efficiently approximate this approach as follows. Generate uniformly distributed $k = 20$ binary numbers each with $s = 8$ bits. The eight bits represent a value from 0 through 255, and correspond to a bin number. The first 8-bit number is the bin number associated with an index 1, the second is the bin number associated with an index of 2, etc. Table 4.2 shows the result of this Monte Carlo simulation using the integer set model. In this case, we generated 10^6 samples. ∎

[2]This assumes that $k \leq 2^{n-s}$

Table 4.1: Registered vector table for an 20-variable index generation function.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	f
0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1
0	0	0	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1	0	0	2
1	0	1	0	1	1	0	1	0	0	0	1	1	0	0	0	0	0	1	1	3
0	0	1	1	0	1	0	1	1	0	0	1	1	0	1	0	1	1	1	0	4
1	0	0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	1	1	1	5
1	1	0	0	1	0	0	1	0	0	1	0	1	0	1	1	0	0	1	1	6
1	0	1	0	1	1	1	1	0	1	1	1	1	1	0	0	0	1	0	1	7
1	0	0	0	1	0	1	1	0	0	1	1	0	0	1	0	1	1	1	1	8
0	1	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	9
0	1	0	1	1	0	0	1	0	1	1	1	0	1	0	0	1	0	0	0	10
1	0	0	1	0	0	1	0	0	0	0	1	0	0	1	1	0	1	0	0	11
0	1	0	0	1	0	0	1	1	1	0	1	1	0	1	0	1	1	0	1	12
0	0	0	0	0	0	1	0	1	1	1	1	0	0	1	0	1	0	1	0	13
0	0	1	1	0	1	1	0	0	1	1	0	0	0	1	1	1	1	1	0	14
0	1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	15
1	0	1	1	1	0	1	0	1	0	0	0	0	0	1	1	1	0	1	0	16
0	0	1	1	0	0	0	0	1	0	1	1	1	0	1	1	1	0	0	0	17
1	0	0	1	1	0	0	0	0	0	1	0	1	0	1	1	0	0	1	1	18
0	0	0	1	1	0	1	1	1	0	1	0	1	1	0	1	1	0	0	0	19
1	1	0	0	1	0	0	0	0	1	1	0	0	1	0	0	1	1	1	1	20

Table 4.2: Number of distinct $(s = 8)$-bit integers.

Rails r	Number of distinct integers μ	Occurrence
4	$13 + 1$	1
4	$14 + 1$	20
4	$15 + 1$	297
5	$16 + 1$	3229
5	$17 + 1$	25749
5	$18 + 1$	129422
5	$19 + 1$	374505
5	$20 + 1$	466777

Table 4.2 shows that, in most cases, the number of distinct integers is either 21 or 20. However, this value can be reduced to $\mu = 14$ *for one case,* $\mu = 15$ *for 20 cases, and* $\mu = 16$ *for 297 cases. This implies that the functions have non-zero probabilities with decompositions where* $\mu \leq 16$*. However, the probability of a decomposition with* $\mu \leq 13$ *is quite low, since a sample set of size* 10^6 *failed to produce a single decomposition with* $\mu \leq 13$*.* ∎

We assume that the functions that appear in the search of the decompositions, and the random set of integers used in a Monte Carlo simulation have similar statistical properties. That is, the probability distribution functions for the functional decompositions are similar to that of random integer sets. The validity of this assumption will be checked in the experiments in Section 6.

5 Procedure to Compute the Column Multiplicity

To validate the use of a Monte Carlo technique in estimating the column multiplicity, we compare the results obtained by a Monte Carlo technique with an exact enumeration in which we enumerate all binary arrays according to the column multiplicity. Every binary array is enumerated exactly once, and, as such, it can be considered a proxy for a 'perfect' Monte Carlo simulation. We use the balls into bins model.

Lemma 5.1 *Assume that there are* t *non-distinct bins and* k *distinct balls. The number of different ways to put* k *distinct balls into* t *non-distinct bins is*

$$S(k,t) = \begin{cases} 1, & \text{if } (t = 1 \text{ or } t = k) \\ S(k-1, t-1) + tS(k-1, t), & \text{otherwise.} \end{cases}$$

Table 5.1: Values for $S(k,t)$

	t								
k	1	2	3	4	5	6	7	8	9
1	1								
2	1	1							
3	1	3	1						
4	1	7	6	1					
5	1	15	25	10	1				
6	1	31	90	65	15	1			
7	1	63	301	350	140	21	1		
8	1	127	966	1701	1050	266	28	1	
9	1	255	3025	7770	6951	2646	462	36	1
10	1	511	9330	34105	42525	22827	5880	750	45

(Proof) The proof is be done by a mathematical induction. $S(k,t)$ can be calculated for three cases:
When $t = 1$: All the balls are in one bin. So, there is only one way.
When $t = k$: All the balls are in k bins. So, there is only one way.
Otherwise: Assume that $k-1$ balls are already in the bins, and the k-th ball is be put into one of the bins. In this case, there exist two cases:
1) The $k-1$ balls are in $t-1$ bins, but one bin is empty. In this case, the number of ways to put the first $k-1$ balls into $t-1$ bins is $S(k-1,t-1)$, by the hypothesis. The k-th ball is put into the empty bin.
2) The $k-1$ balls are in t bins. No bin is empty. In this case, the number of ways to put the first $k-1$ balls into t bins is $S(k-1,t)$, by the hypothesis. Also, there are t ways to put the k-th ball into one of t bins. So, the total number of ways is $tS(k-1,t)$. From these, we have the lemma. □

Note that $S(k,t)$ is the **Stirling number of the second kind** [7].

Example 5.1 *Table 5.1 shows the values of* $S(k,t)$ *for* $k \leq 10$ *and* $t \leq 9$.

Theorem 5.1 *Consider the binary arrays that have* 2^s *columns and* k *rows. The number of arrays with* t *distinct columns is*

$$c_{k,t} = P(2^s, t)S(k,t),$$

where $S(k,t)$ *is the Stirling number of the second kind, and* $P(n,r) = \frac{n!}{(n-r)!}$.

(Proof) The number of ways to permute t distinct patterns out of 2^s distinct patterns is $P(2^s, t)$. □

To validate the Monte Carlo technique, we ran the simulation for $s = 3$ and $k = 8$ for a total of $(2^s)^k = 2^{24} = 16777216$ samples, which is the number of 3×8 binary arrays. We also used the exact enumeration using Theorem 5.1. Table 5.2 compares the results. This shows that there is a close correlation between the Monte Carlo simulation and exact enumeration.

For small k, we can pre-compute the table of the Stirling numbers, and store it in the hard disk to compute $c_{k,t}$. However, for large k, the table becomes too large. Thus, we use the Monte Carlo approach for large k.

Table 5.2: Comparison of the Monte Carlo technique with exact enumeration ($s = 3, k = 8$).

# of Distinct Columns	Monte Carlo	Theorem 5.1
1	4	8
2	7162	7112
3	326013	324576
4	2857425	2857680
5	7053129	7056000
6	5362312	5362560
7	1130883	1128960
8	40288	40320
Total	16777216	16777216

6 Experimental Results

6.1 Decompositions of 20-Variable Functions

We assume that the distribution of the column multiplicities during decompositions is similar to the random integer sets used in the Monte Carlo simulation.

To confirm this, we generated 10 sample index generation functions of $n = 20$ and $k = 20$. Then, for each function, we counted the column multiplicities for all the decom-

positions, where the number of bound variables is $s = 8$. Note that the number of ways to select 8 variables out of 20 variables is $\binom{20}{8} = 125970$.

Table 6.1 summarizes the distributions of column multiplicities. The first column shows the number of rails $r = \lceil \log_2 \mu \rceil$. The second column shows the column multiplicity μ. The third to 12th columns show the distributions for f_1 to f_{10}. For example, in f_1, the minimum column multiplicity is $\mu = 15 + 1$, and 20 decompositions produce this μ. For f_5, the minimum multiplicity is $\mu = 13 + 1$, and 10 decompositions produce this μ. For f_9, the minimum multiplicity is $\mu = 14 + 1$, and 18 decompositions produce this μ. For the other 8 functions, the minimum multiplicities are $\mu = 15 + 1$. The column headed with *SUM* denotes the sum of 10 sample functions. The rightmost column, headed with *Monte* denotes the result of the Monte Carlo simulation using the integer set model. The number of sample sets generated for the simulation is $10 \times \binom{20}{8} = 1259700$, which is equal to the total number of decompositions for 10 sample functions.

Table 6.1 shows that, for all 10 sample functions, the column multiplicities can be reduced to $\mu = 15 + 1$ or less. This means that the number of rails r can be reduced from five to four. The Monte Carlo simulation shows that, for $374 + 25 + 2 = 401$ combinations out of 1259700, the column multiplicities are reduced to $\mu = 15 + 1$ or less. This shows that there is an incentive to find a decomposition with small column multiplicity, but it may be hard to find.

Table 6.1: Decompositions of 20-variable index generation functions with $k = 20$, $(s = 8)$.

r	μ	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	SUM	$Monte$
4	$13+1$	0	0	0	0	10	0	0	0	0	0	10	2
4	$14+1$	0	0	0	0	181	0	0	0	18	0	199	25
4	$15+1$	20	18	27	7	1338	5	24	3	138	9	1589	374
5	$16+1$	457	571	353	164	5478	144	366	122	1237	172	9064	4101
5	$17+1$	4673	4775	3346	1669	14934	1761	2645	1683	6413	2242	44141	32505
5	$18+1$	22084	20615	18172	10841	28903	11668	14770	13178	23163	13887	177281	163015
5	$19+1$	48736	49356	49694	41775	40076	42141	46962	47846	49122	46922	462630	471686
5	$20+1$	50000	50635	54378	71514	35050	70251	61203	63138	45879	62738	564786	587992

Table 6.2: Decomposition of a 64-variable index generation function with $k = 20$ $(s = 8)$.

r	μ	$Occurrence$	$Monte$
4	$11+1$	0	1
4	$12+1$	20	73
4	$13+1$	971	2768
4	$14+1$	33301	69099
4	$15+1$	759722	1196778
5	$16+1$	11265390	14274472
5	$17+1$	103232971	113605327
5	$18+1$	562995509	574201989
5	$19+1$	1675277777	1656555761
5	$20+1$	2072599707	2066259100
	$Total$	4426165368	4426165368

6.2 Decomposition of a 64-Variable Function

In the previous experiment, the number of variables was only 20, and the number of the decompositions was $\binom{20}{8} = 125970$.

In this part, we used a sample function with $n = 64$ and $k = 20$. As before, the number of bound variables is $s = 8$. Note that, the number of decompositions is now $\binom{64}{8} = 4426165368$. Table 6.2 shows the results. The first column shows the number of rails r; the second column shows the column multiplicities μ; the third column shows the number of decompositions that produced the corresponding μ. The rightmost column headed by $Monte$ shows the number of occurrences in the Monte Carlo simulation. In the Monte Carlo simulation, the number of possible ways to generate sets of 20 integers of 8 bits is $(2^8)^{20} = 2^{160} \simeq 1.46 \times 10^{48}$. In this experiment, we generated $\binom{64}{8} = 4426165368$ sample sets.

In this example, the Monte Carlo method provides a good approximation when μ is large, but not so good when μ is small. Note that the CPU time for the Monte Carlo simulation was about 10 minutes, while that for the exhaustive decomposition search was 22 minutes.

With the Monte Carlo simulation, we can see that the probability of finding a decomposition with $\mu = 16$ (i.e., $r = 4$ rails) is quite high if the good solutions are distributed uniformly, and if we try more than 10^4 samples randomly. However, the probability that with $\mu \leq 8$ (i.e., $r = 3$ rails) is almost zero. Thus, once we find a decomposition with $\mu \leq 16$, we can stop the search; it is not likely that we will find a decomposition with a smaller r.

7 A Method To Assess Programmable Architecture

Problem 1 *Design a programmable architecture for an index generation function with* $n = 500$ *and* $k = 100$ *using a pair IGUs.*

(Solution) In a decomposition, the number of the rails is at most $q = \lceil \log_2(k+1) \rceil = 7$. A Monte Carlo simulation with $s = 11$ and $k = 100$ shows that the minimum column multiplicity of the decompositions among 10^6 samples is $\mu = 87$. Since, the number of rails is $r = \lceil \log_2 \mu \rceil = 7$, the function can be realized by a cascade as shown in Fig. 7.1. IGU1 has 255 inputs and 7 outputs, while IGU2 has $7 + 245 = 252$ inputs and 7 outputs.

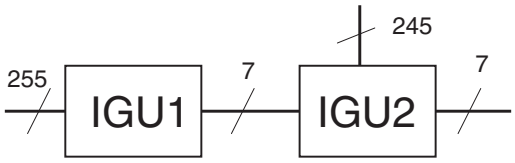

Figure 7.1: Architecture using two IGUs.

Problem 2 *Design a programmable architecture for an index generation function with* $n = 20$ *and* $k = 68$ *using a pair of LUTs.*

(Solution) In a decomposition, the number of rails is at most $q = \lceil \log_2(k+1) \rceil = 7$, by Lemma 3.1. Let the number of bound variables be $s = 10$. The Monte Carlo simulation with $s = 10$ and $k = 68$ shows that the minimum column multiplicity of the decompositions among 10^6 samples is $\mu = 57$. Thus, the number of rails is reduced to $\lceil \log_2 \mu \rceil = 6$. The function can be realized by a cascade as shown in Fig. 7.2. LUT1 has 10 inputs and 6 outputs, while LUT2 has $6 + 10 = 16$ inputs and 7 outputs. Since LUT2 has 16 inputs, and is much larger than LUT1, the circuit is not so efficient.

So, we increase the number of inputs to LUT1 to $s = 12$. A Monte Carlo simulation with $s = 12$ and $k = 68$ shows that the minimum column multiplicity of the decompositions among 10^6 samples is $\mu = 62$. Thus, the number of rails is still $\lceil \log_2 \mu \rceil = 6$. The function can be realized by a cascade as shown in Fig. 7.3, which is more efficient than Fig. 7.2.

Next, we further increase the number of inputs to LUT1 to $s = 14$. A Monte Carlo simulation with $s = 14$ and $k = 68$ shows that the minimum column multiplicity of the decompositions among 10^6 samples is $\mu = 65$. Thus, the number of rails is increased to $\lceil \log_2 \mu \rceil = 7$, as shown in Fig. 7.4, which is less efficient than Fig. 7.3.

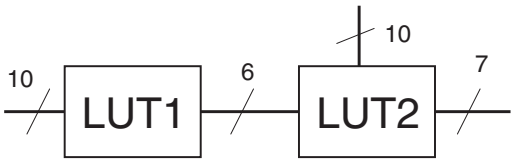

Figure 7.2: Architecture using two LUTs.

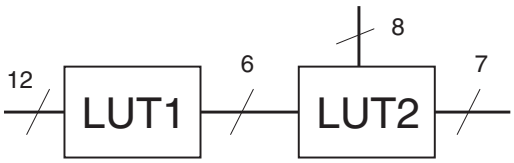

Figure 7.3: Architecture using two LUTs.

8 Conclusion and Comments

In this chapter, we present a Monte Carlo method to predict the column multiplicity of the decomposition charts for random index generation functions. We also show a procedure to compute the multiplicities of decomposition charts. Comparison with the exact enumerations shows that the Monte Carlo method using integer model produces good approximations to exact enumeration.

When we design a programmable architecture for index generation functions, in many cases, we know only the numbers of inputs and registered vectors, but not the detail of the functions. In such cases, the method to assess the programmable architecture presented in this chapter is quite useful.

A different, but related problem is to find decompositions for specific index generation functions. We developed a heuristic [16] and exact [17] algorithms to find decompositions of index generation functions. In a functional decomposition algorithm, when a decomposition with a column multiplicity μ_1 is found, the next step is to find a decomposition with a column multiplicity μ_2, where $\lceil \log_2 \mu_1 \rceil < \lceil \log_2 \mu_1 \rceil$. If there is no solution, we can stop the search. Analysis of column multiplicities for index generation functions were useful to find these algorithms.

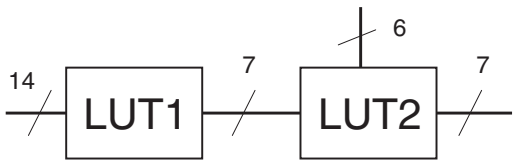

Figure 7.4: Architecture using two LUTs.

Acknowledgments

This research is partly supported by the Japan Society for the Promotion of Science (JSPS) Grant in Aid for Scientific Research. Discussion with Mr. Kyu Matsuura was useful to improve Section 5. Also, the reviewers' comments improved the presentation of the chapter.

References

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

[2] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," *IEEE/ACM International Conference on Computer-Aided Design*, (ICCAD-1997), pp. 78-82, Nov. 1997.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.

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

[5] V. Kravets and K. Sakallah, "Constructive library-aware synthesis using symmetries," *Design, Automation and Test in Europe*, (DATE-2000), March 2000, pp. 208-213, Paris, France.

[6] Y-T. Lai, M. Pedram and S. B. K. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis,"*30th ACM/IEEE Design Automation Conference*, (DAC-1993), June 1993, pp. 642-647.

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

[8] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition,"*Synthesis And System Integration of Mixed Technologies*, (SASIMI-1998), pp. 44-50, Oct. 1998.

[9] H. Nakahara, T. Sasao, M. Matsuura, H. Iwamoto, and Y. Terao, "A memory-based IPv6 lookup architecture using parallel index generation units," *IEICE Trans. Inf. and Syst.* Vol. E98-D, No. 2, pp. 262-271, Feb., 2015.

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

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

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

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

[14] T. Sasao, "Index generation functions: Tutorial,"*Journal of Multiple-Valued Logic and Soft Computing*, Vol. 23, No. 3-4, pp. 235-263, 2014.

[15] T. Sasao, "A realization of index generation functions using multiple IGUs," *Inter. Symp. on Multiple-Valued Logic*, (ISMVL-2016), Sapporo, Japan, May 17-19, 2016, pp.113-118.

[16] T. Sasao, K. Matsuura, and Y. Iguchi, "A heuristic decomposition of index generation functions with many variables," *The 20th Workshop on Synthesis And System Integration of Mixed Information Technologies* (SASIMI-2016), Kyoto, Oct. 24, 2016, R1-6, pp. 23-28.

[17] T. Sasao, K. Matsuura, and Y. Iguchi, "An algorithm to find optimum support-reducing decompositions for index generation functions." *Design, Automation and Test in Europe*, (DATE-2017), March 27-31, 2017, Lausanne, Switzerland.