Compatible Class Encoding in Roth-Karp Decomposition for Two-Output LUT Architecture

Juinn-Dar Huang, Jing-Yang Jou and Wen-Zen Shen

Department of Electronics Engineering,
National Chiao Tung University,
Hsinchu, Taiwan, R.O.C.

Abstract

Roth-Karp decomposition is one of the most popular techniques for LUT-based FPGA technology mapping because it can decompose a node into a set of nodes with fewer numbers of fanins. In this paper, we show how to formulate the compatible class encoding problem in Roth-Karp decomposition as a symbolic-output encoding problem in order to exploit the feature of the two-output LUT architecture. Based on this formulation, we also develop an encoding algorithm to minimize the number of LUT's required to implement the logic circuit. Experimental results show that our encoding algorithm can produce promising results in the logic synthesis environment for the two-output LUT architecture.

1. Introduction

Field Programmable Gate Arrays (FPGA's) are modern logic devices which can be programmed by the users to implement their own logic circuits. Because of the short turnaround time compared with that of the standard ASIC process, they become increasingly popular in rapid system prototyping recently. Many FPGA architectures have been proposed and the Look-Up Table(LUT)-based architecture is the most popular one. It consists of many configurable LUT's and each LUT can implement any *k*-input function. Besides, there is another similar LUT-based FPGA architecture which implements not only single-output functions but also two-output functions. A LUT of this architecture can implement either one *k*-input function or two (*k-1*)-input functions with totally *k* inputs. For example, in Xilinx XC3000 architecture[1], *k* is equal to 5.

Many algorithms developed for LUT-based FPGA technology mapping have been proposed in previous studies[2-11]. Most of these algorithms first decompose the given Boolean network to be *k*-feasible. A Boolean network is said to be *k*-feasible if all nodes in the network are *k*-feasible, and a node is said to be *k*-feasible if the number of its fanins is no more than *k*. Hence, the corresponding circuit can be directly realized by an one-to-one mapping between nodes and LUT's. If there are some nodes that are not *k*-feasible, they then need to be decomposed to be a set of *k*-feasible nodes. Many proposed decomposition techniques, such as AND-OR decomposition, cofactoring, disjoint decomposition, if-then-else DAG, communication complexity reduction[12], and Roth-Karp decomposition[13], are widely used here. In this paper, we only focus on Roth-Karp decomposition.

Given the LUT-based FPGA as the target architecture, two interesting problems in Roth-Karp decomposition should be noticed:

1. How to select the lambda set?
2. How to encode the compatible classes?

The algorithm proposed in [11] provides a heuristic to choose a good lambda set. Another algorithm proposed in [10] formulates Problem 2 as a symbolic-input encoding problem. However, both of these two algorithms only consider the single-output LUT architecture. In this paper, we propose a new formulation for Problem 2 and develop a new compatible class encoding algorithm which can fully exploit the feature of the two-output LUT architecture.

This paper is organized as follows. Section 2 describes the compatible class encoding problem in Roth-Karp decomposition. In Section 3, our new encoding algorithm which addresses the two-output LUT architecture is given in detail. Section 4 shows experimental results and the concluding remarks are given in Section 5.

2. Compatible Class Encoding

Definition 2.1:

Let X and Y be two sets of binary variables, $X \cap Y = \varnothing$; $B^{|x|}$ and $B^{|y|}$ are Cartesian products spanned by X and Y, respectively. Then, given a completely specified function $F: B^{|x|} \times B^{|y|} \rightarrow B$, we say that $x_1, x_2 \in B^{|x|}$ are **compatible** with respect to F, denoted as $x_1 \sim x_2$, if $\forall\, y \in B^{|y|}$, (x_1, y) and $(x_2, y) \in B^{|x|} \times B^{|y|}$ such that $F(x_1, y) = F(x_2, y)$. □

Each element $\in B^{|x|}$ is called a λ **minterm**. All mutually compatible λ minterms can be grouped together to form a **compatible class**, and all compatible classes are pairwisely disjoint.

Theorem 2.1:

Given two functions $\vec{\alpha} : B^{|x|} \rightarrow W$ and $G: W \times B^{|y|} \rightarrow B$, such that

$$\forall\, (x, y) \in B^{|x|} \times B^{|y|},\ F(x, y) = G(\vec{\alpha}(x), y) \qquad (1)$$

holds if and only if

$$\forall\, x_1, x_2 \in B^{|x|},\ \vec{\alpha}(x_1) = \vec{\alpha}(x_2) \Rightarrow x_1 \sim x_2 \qquad (2)$$ □

$\vec{\alpha}$ is a function with binary inputs and a symbolic output. X is called the **bound** (λ) **set**. Y is called the **free** (μ) **set**.

Property 2.1:

The number of the admissible values in W, $|W|$, must be no less than the number of compatible classes in X. □

From Property 2.1, minimum $|W|$ is equal to the number of compatible classes in X and can be obtained by redefining Eq. (2) as:

$$\forall\, x_1, x_2 \in B^{|x|},\ \vec{\alpha}(x_1) = \vec{\alpha}(x_2) \Leftrightarrow x_1 \sim x_2 \qquad (2')$$

In order to implement $\vec{\alpha}$ by binary logic, the symbolic-output encoding for W has to be performed. At least $t = \lceil log_2^{|W|} \rceil$ binary-output functions, α_1, α_2, ..., and α_t, are required to encode $\vec{\alpha}$. Hence, Eq. (1) can be rewritten as:

$$F(x, y) = G'(\alpha_1(x), \alpha_2(x), ..., \alpha_t(x), y) \quad (1')$$

It is clear that the disjoint Roth-Karp decomposition can be used to reduce the number of fanins of a function under the condition of $t < |X|$.

From the above discussion, we find that Roth-Karp decomposition only requires that λ minterms belonging to the same compatible class are encoded with the same code, i.e., if $x_1 \sim x_2$, then $\vec{\alpha}(x_1) = \vec{\alpha}(x_2)$. In other words, to have a correct Roth-Karp decomposition, we only need to assign a unique code to each compatible class and do not have to care what the code is. However, different encoding combinations for the compatible classes will result in different α_1, α_2, ..., α_t, and G'.

There is a compatible class encoding strategy being proposed in [10]. It models the compatible class encoding problem as the classical symbolic-input encoding problem. Many existing techniques are then used to encode the compatible classes for minimizing the literal counts of G'. Because it assumes that the better decomposition quality can be obtained if the resulting G' is simple, i.e., has smaller number of literals. However, this strategy only concentrates on single-output functions. Furthermore, it did not properly take advantage of the feature of either the single-output or the two-output LUT architecture.

To exploit the property of the two-output LUT architecture, we formulate the compatible class encoding problem as:

To find a set of compatible class encoding patterns to encode as many α functions to be independent of at least one of their input variables as possible.

Thus, two α functions which are independent of at least one of their input variables can be merged into a two-output k-LUT. In fact, our formulation of this encoding problem is a kind of the symbolic-output encoding because we encode the symbolic output variable W into binary-output encoding functions α_1, α_2, ..., and α_t.

3. Our Compatible Class Encoding Algorithm

For the easy illustration, the LUT used in this section is either a 4-LUT or a two-output 4-LUT.

Example 1:

Given a compatible class encoding function $\vec{\alpha}$ with a set of inputs X as the λ set and a symbolic output variable W:

$\vec{\alpha} : B^{|x|} \rightarrow W$, where $X = \{x_1, x_2, x_3, x_4\}$ and $W = \{0,1,2,3,4\}$

x_3x_4 \ x_1x_2	00	01	11	10
00	2	1	1	2
01	4	2	0	3
11	1	2	2	1
10	0	3	1	2

W

The admissible values of W are defined to be the id's of the compatible classes. Thus, $\vec{\alpha}$ maps each λ minterm to the id of the compatible class it belongs. In this example, 3 ($\lceil log_2^5 \rceil$) binary-output functions α_1, α_2, and α_3, are needed to implement $\vec{\alpha}$. Suppose these symbolic values are randomly encoded without any strategy; for instance, each symbolic value is encoded with its binary bit pattern. Then, the encoding of $\vec{\alpha}$ and the resulting Boolean functions are shown below:

class id	α_3	α_2	α_1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	0	0	0
10	0	0	0	0

$\alpha_3 = \bar{x}_1\bar{x}_2\bar{x}_3x_4$

x_3x_4 \ x_1x_2	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	0	1	1	0
10	0	1	0	1

x_3x_4 \ x_1x_2	00	01	11	10
00	0	1	1	0
01	0	0	0	1
11	1	0	0	1
10	0	1	1	0

$\alpha_2 = \bar{x}_2\bar{x}_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_4 + \bar{x}_1x_2x_3 + \bar{x}_1x_2x_4 + x_2x_3x_4$

$\alpha_1 = x_2\bar{x}_4 + x_1\bar{x}_2x_4 + \bar{x}_2x_3x_4$

Obviously, three LUT's are required to implement these α functions since all of them depend on all four input variables.

Example 2:

As in Example 1, a better encoding of $\vec{\alpha}$ is used here. The encoding of $\vec{\alpha}$ and the resulting Boolean functions are shown below:

class id	α_3	α_2	α_1
0	1	1	0
1	0	0	0
2	1	0	0
3	0	1	0
4	0	0	1

x_3x_4 \ x_1x_2	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

$\alpha_3 = x_2x_4 + \bar{x}_2\bar{x}_4$

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	0	0	0
10	1	1	0	0

x_3x_4 \ x_1x_2	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	0	0	0
10	0	0	0	0

$\alpha_2 = x_1\bar{x}_3x_4 + \bar{x}_1x_3\bar{x}_4$

$\alpha_1 = \bar{x}_1\bar{x}_2\bar{x}_3x_4$

We find that α_3 is independent of x_1 and x_3, and α_2 is independent of x_2, respectively. Therefore, α_3 and α_2 can be merged into a two-output LUT. Two instead of three LUT's are thus required to implement these α functions.

Now we give some definitions and derive some properties from them. Based on these definitions and properties, our compatible class encoding algorithm is then constructed.

Definition 3.1:

An **independent set** with respect to an input variable *x*, denoted as ***ISx***, is defined as a set of class id's such that there exists a binary-input/output function α being independent of *x* where

the ON-set of α, $\alpha^{ON} = \{ m \mid m \text{ is a } \lambda \text{ minterm where } \bar{\alpha}(m) \in ISx \}$. □

Example 3:

In Example 1, {0, 2} is an ISx_1. Because we can find a function $\alpha_3 = x_2x_4 + \bar{x}_2\bar{x}_4$ as illustrated in Example 2 satisfies Definition 3.1.

Definition 3.2:

An *ISx* is a **minimum independent set** with respect to the input variable *x*, denoted as ***MISx***, if and only if

$\forall T$, $T \subset ISx$ and $T \neq \varnothing \Rightarrow T$ is no longer an *ISx*. □

Example 4:

In Example 1, {0, 2} and {1, 3, 4} are two $MISx_1$'s by Definition 3.2.

Property 3.1:

Given two λ minterms *m* and *m'* where *m'* is the same with *m* except that it is complemented at the position of variable *x*. Let $\bar{\alpha}(m) = k$ and $\bar{\alpha}(m') = k'$, then *k* and *k'* must belong to the same *MISx* by Definition 3.1. □

Property 3.2:

Two *MISx*'s are either identical or disjoint, and the union set of all *MISx*'s is *W*. □

From Property 3.1 and 3.2, we can easily develop an algorithm to find all *MIS*'s for a given variable *x*.

Example 5:

The set, ***Set_MISx***, containing all *MIS*'s with respect to each variable *x* of Example 1 is shown below, respectively.

$Set\_MIS_{x_1} = \{\{0,2\},\{1,3,4\}\}$ $Set\_MIS_{x_3} = \{\{0,2\},\{1,3,4\}\}$

$Set\_MIS_{x_2} = \{\{0,3\},\{1,2,4\}\}$ $Set\_MIS_{x_4} = \{\{0,1,2,3,4\}\}$

Property 3.3:

Each combination of an arbitrary number of *MISx*'s represents an *ISx*. So the size of the set ***Set_ISx*** containing all *ISx*'s is equal to $2^{|Set\_MIS_x|}$. □

Thus, *Set_ISx* can be derived from *Set_MISx* by applying Property 3.3. For example, Set_ISx_1 in Example 1 is {∅, {0, 2}, {1, 3, 4}, {0, 1, 2, 3, 4}}. Notice that ∅ and {0, 1, 2, 3, 4} (*W*) are discarded since they are useless for encoding purpose described later. At this point, we introduce a terminology, **dichotomy**, which is first used for symbolic-input encoding in [14]. The notion of dichotomy is slightly modified here for convenience.

Definition 3.3:

A **dichotomy** with respect to *x*, ***Dx***, is given by an ordered pair, denoted as (*l*:*r*), where *l* is an *ISx* and *r* is *W*- *ISx*. □

Definition 3.4:

Two dichotomies, $D_1 = (l_1{:}r_1)$ and $D_2 = (l_2{:}r_2)$, are **equivalent** if ($l_1 = l_2$ and $r_1 = r_2$) or ($l_1 = r_2$ and $r_1 = l_2$). □

From Definition 3.3 and 3.4, a set of distinct, i.e., non-equivalent, dichotomies with respect to *x*, ***Set_Dx***, can be generated from *ISx*. For instance, Set_Dx_1 in Example 1 is {(02:134)}. After discarding two useless *ISx*'s (∅ and *W*) and exploiting the equivalence relation among dichotomies, the size of *Set_Dx* is given by Property 3.4:

Property 3.4:

$$|Set\_D_x| = \frac{|Set\_IS_x| - 2}{2} = \frac{2^{|Set\_MIS_x|} - 2}{2}$$ □

Property 3.5:

A dichotomy $D_x = (l{:}r)$ can be used to make the encoding function α_i independent of *x* if the *i*-th bit of the code of the compatible class *c* is equal to:

$$\begin{cases} 1 & \text{if the id of } c \in l, \\ 0 & \text{if the id of } c \in r. \end{cases}$$ □

Example 6:

In Example 1, there exists a dichotomy, $D = (02{:}134) \in Set\_Dx_1$. If α_3 is encoded by *D*:

class id	α_3	α_2	α_1
0	1	-	-
1	0	-	-
2	1	-	-
3	0	-	-
4	0	-	-

x_3x_4 \ x_1x_2	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

$\alpha_3 = x_2x_4 + \bar{x}_2\bar{x}_4$

It is clear that α_3 is independent of x_1 under this encoding pattern.

Because we do not care which variable *x* that α_i is independent of, dichotomies from different *Set_Dx*'s can be merged into a set ***Set_D***. Then, we formulate the compatible class encoding problem to satisfy the following three encoding constraints:

1. Each compatible class must be encoded with a distinct code.
2. Only minimum number(*t*) of encoding bits are allowed to be used.
3. Use as many dichotomies for the encoding as possible.

Constraint 1 is given to satisfy the definition of Roth-Karp decomposition. Constraint 2 is given to minimize the number of LUT's used to implement α functions. Constraint 3 is given to exploit the feature of the two-output LUT architecture.

While merging *Set_Dx*'s into *Set_D*, if two equivalent dichotomies, D_1 and D_2, are from two different Set_Dx_1 and Set_Dx_2, respectively, then α_i encoded by either D_1 or D_2 is independent of both x_1 and x_2. In this case, the number of fanins of α_i can be further reduced and the number of interconnection nets required for routing is also reduced. Therefore, a dichotomy which is independent of the most inputs should be chosen for encoding first.

A procedure, *Exhaustive_Search_Encoding*, is developed to encode the compatible classes under three constraints described above. It first tries to find the maximum number(*t*) of dichotomies for encoding to satisfy Constraint 1 and 2. If it fails, it reduces the number of encoding dichotomies by 1 and tries again. This procedure guarantees to get the optimum encoding solution, i.e., the maximum number of dichotomies can be found to encode α functions without violating Constraint 1 and 2.

Example 7:

By reexamining Example 1, we find that two dichotomies (02:134) and (03:124) can be found by *Exhaustive_Search_Encoding* to encode α_3 and α_2, respectively. α_1 is then encoded to satisfy Constraint 1. Therefore, the encoding result is identical with that illustrated in Example 2.

The time efficiency of this procedure is not that good. Suppose there are d dichotomies being used to encode t bits, then the worst case, in which no one can be used for encoding, takes $C_t^d + C_{t-1}^d + \cdots + C_1^d$ iterations when $d > t$. We find that the number of iterations dramatically increases as d increases. Thus, a more efficient algorithm should be developed for the practical usage. Consider an example that the numbers of the compatible classes and dichotomies are 6 and 5, respectively. t is then equal to 3 in this case. The search space of finding the optimum encoding solution for this example is shown in Fig. 1. Any path from node *S* to the nodes of the third level in the tree represents a possible combination of three dichotomies. Similarly, any path from node *S* to the nodes of the second level in the tree represents a possible combination of two dichotomies.

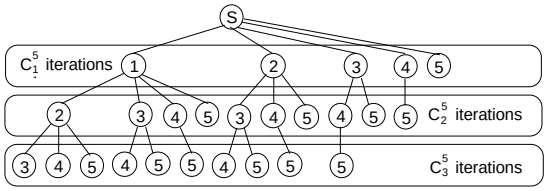

Fig. 1 : The search space of finding the maximum number of dichotomies for the encoding.

In this example, 25 iterations are required to find the optimum solution in the worst case. Suppose that the first dichotomy is (0:12345), we find that it is impossible to distinguish class 1~5 no matter what the remaining dichotomies are because only two more bits are allowed to use. Thus, the subtree rooted at the first dichotomy can be pruned without affecting the search of the optimum solution. This pruning algorithm is illustrated in Fig. 2.

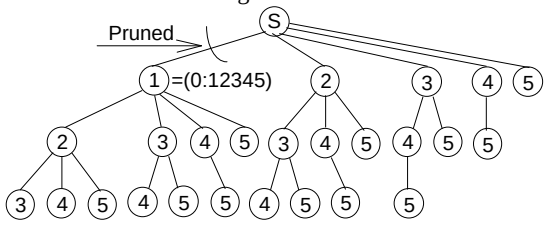

Fig. 2 : An example of the pruning strategy on the search space of the dichotomies.

Hence, the pruning algorithm can be formally described as:

At any node of the search space of the dichotomies, if all the compatible classes cannot be partitioned into sets whose sizes are no more than $2^{|remaining\ bits|}$ *by using the current dichotomy, then the subtree rooted at this node is pruned.*

This pruning algorithm also guarantees to get the optimum encoding solution and executes more efficiently than the procedure *Exhaustive_Search_Encoding*.

4. Experimental Results

The algorithm described above has been implemented in SIS environment which is developed by UC Berkeley[15]. Besides, our algorithm is integrated into a version of Roth-Karp decomposition[11], which has the lambda set selection strategy, to enhance its performance. In order to investigate the quality of our encoding algorithm, denoted as Algorithm 3, two experiments are conducted over a large set of MCNC and ISCAS benchmark circuits to compare the results with those of another two versions of Roth-Karp decomposition. Algorithm 1 has neither the lambda set selection strategy nor the compatible class encoding strategy and is used in mis-pga[3]. Algorithm 2 has only the lambda set selection strategy and is implemented in [11]. The target architecture is the Xilinx XC3000 FPGA which can implement either one 5-input function or two 4-input functions with totally 5 inputs as described above. The initial networks of one experiment are two-level circuits and are obtained by performing the SIS script:

```
collapse
simplify -d -m nocomp
```

The initial networks of another experiment are multi-level circuits and are obtained by performing the SIS standard multi-level optimization script. After obtaining the initial networks, the same mapping script:

```
xl_k_decomp   /*various algorithms applied here */
xl_partition  -tm
xl_cover
xl_merge      -l
```

is used in both experiments. This script first decomposes the network to be 5-feasible. Algorithm 1, 2, and 3 are applied here. Thus, each node can be implemented by a 5-LUT. Then, it tries to merge pairs of nodes which can be implemented by two-output 5-LUT's as many as possible. Both experiments run on a SUN SPARC 5 workstation and the results are shown in Table I and II, respectively.

Table I shows the results on 16 two-level Benchmarks. On average, Algorithm 3 requires 41% and 20% fewer LUT's than that of Algorithm 1 and 2, respectively. Moreover, Algorithm 3 is also very time-efficient. It only requires 14% and 90% CPU time than that of Algorithm 1 and 2, respectively. Table II shows the results on 26 multi-level benchmarks. We find that Algorithm 2 and 3 produce almost the same results. On average, they both require 32% fewer LUT's than that of Algorithm 1, and also take 15~20% less CPU time than that of Algorithm 1 in this experiment.

These two experiments show that our encoding strategy can provide greater improvement for circuits starting with two-level forms but no significant improvement for circuits starting with multi-level forms. It is because that node functions of the multi-level circuits optimized by SIS are simpler and have fewer inputs than node functions of the two-level circuits. The similar reason has also been suggested in [10]. Therefore, our encoding strategy does not make much difference for multi-level circuits after choosing a good lambda set for decomposing functions.

5. Conclusions

In this paper, we discuss the compatible class encoding problem in Roth-Karp decomposition for two-output LUT

architecture. We show how to formulate this problem as a symbolic-output encoding problem. Based on this formulation, we also develop an encoding algorithm and integrate it into a version of Roth-Karp decomposition with the lambda set selection strategy. Experiment results show that our new encoding algorithm can efficiently use fewer LUT's to implement circuits starting with two-level forms for the two-output LUT architecture. By investigating the optimization strategy of current logic synthesis systems, our new encoding technique is very useful in both two-level and multi-level logic synthesis systems targeting for the two-output LUT-based FPGA.

References

[1] Xilinx Inc., 2100, Logic Drive, San Jose, CA-95124, *The Programmable Logic Data Book*.

[2] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," in *Proc. 27th Design Automation Conf.*, June 1990, pp.620-625.

[3] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp.564-567.

[4] R. J. Francis, J. Rose, and K. Chung, "Chortle : A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays," in *Proc. 27th Design Automation Conf.*, June 1990, pp.613-619.

[5] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf : Fast Technology Mapping for Lookup Table-Based FPGA's," in *Proc. 28th Design Automation Conf.*, June 1991, pp.227-233.

[6] K. Karplus, "Xmap : A Technology Mapper for Table-Lookup Field Programmable Gate Arrays," in *Proc. 28th Design Automation Conf.*, June 1991, pp.240-243.

[7] N. Woo, "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," in *Proc. 28th Design Automation Conf.*, June 1991, pp.248-251.

[8] D. Filo, J. C. Yang, F. Mailhot, and G. D. Micheli, "Technology Mapping for a Two-Output RAM-based Field-Programmable Gate Arrays," in *Proc. European Design Automation Conf.*, Feb. 1991, pp.534-538.

[9] Y. T. Lai, M. Pedram, and Sarma B. K. Vrudhula, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," in *Proc. 30th Design Automation Conf.*, June 1993, pp.642-647.

[10] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum Functional Decomposition Using Encoding," in *Proc. 31st Design Automation Conf.*, June 1994, pp.408-414.

[11] W.-Z. Shen, J.-D. Huang, and S.-M. Chao, "Lambda Set Selection in Roth-Karp Decomposition for LUT-Based FPGA Technology Mapping," in *Proc. 32nd Design Automation Conf.*, June 1995, pp.65-69.

[12] TingTing Hwang, Robert M. Owens, Mary J. Irwin, and Kuo Hua Wang, "Logic Synthesis for Field-Programmable Gate Arrays," in *IEEE Trans. on Computer-Aided Design*, Oct. 1994, pp.1280-1287.

[13] J. P. Roth, and R. M. Karp, "Minimization Over Boolean Graphs," in *IBM Journal of Research and Development*, April 1962, pp.227-238.

[14] S. Yang and M. Ciesielski, "Optimum and Suboptimal Algorithm for Input Encoding and Its Relationship to Logic Minimization," in *IEEE Trans. on Computer-Aided Design*, Jan. 1991, pp.4-12.

[15] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS : A Multi-Level Logic Optimization System," in *IEEE Trans. on Computer-Aided Design*, Nov. 1987, pp.1062-1081.

Table I : The experimental results of two-level circuits.

CKT name	#in	#out	Algorithm 1		Algorithm 2		Algorithm 3	
			#LUT	time	#LUT	time	#LUT	time
5xp1	7	10	17	3.8	15	2.6	15	1.6
9sym	9	1	7	2.2	7	3.5	7	4.7
alu2	10	6	109	86.7	74	27.3	54	23.4
apex4	9	19	895	1827.6	409	125.8	393	132.1
b9	41	21	99	28.2	82	22.0	54	18.6
clip	9	5	70	28.8	33	9.2	24	13.4
count	35	16	78	22.4	50	30.4	50	29.3
duke2	22	29	711	1521.5	661	231.6	339	167.8
e64	65	65	528	39.2	520	62.9	520	65.1
f51m	8	8	20	1.7	12	1.8	12	2.7
misex1	8	7	14	1.7	13	3.1	11	2.4
misex2	25	18	41	5.7	36	4.6	34	5.7
rd73	7	13	7	2.3	7	4.0	7	3.5
rd84	8	4	12	5.5	12	11.7	12	12.7
sao2	10	14	50	11.1	32	6.3	32	9.1
z4ml	7	14	8	1.3	5	1.0	5	0.7
Total			2666	3589.7	1968	547.8	1569	492.8
Normalized 1			1	1	0.74	0.15	0.59	0.14
Normalized 2			1.35	6.55	1	1	0.80	0.90

Table II : The experimental results of multi-level circuits.

CKT name	#in	#out	Algorithm 1		Algorithm 2		Algorithm 3	
			#LUT	time	#LUT	time	#LUT	time
5xp1	7	10	20	3.1	19	2.2	19	2.7
9sym	9	1	93	23.9	57	19.8	57	21.2
9symml	9	1	79	28.8	60	17.2	58	17.1
alu2	10	6	161	73.5	98	49.5	98	50.3
alu4	14	8	303	45.6	179	27.1	180	31.7
apex6	135	99	181	11.7	186	15.1	186	22.1
apex7	49	37	48	8.8	51	11.0	51	10.2
b9	41	21	30	5.2	29	4.5	29	4.2
bw	5	28	40	16.9	47	13.3	47	11.5
C499	41	32	62	14.1	62	13.5	62	14.2
C880	60	26	134	43.8	84	22.6	84	23.4
clip	9	5	34	6.7	24	6.3	25	5.8
count	35	16	27	3.5	27	2.7	27	3.0
des	256	245	1539	355.3	871	239.3	865	268.2
duke2	22	29	153	55.7	103	31.9	105	33.0
e64	65	65	56	7.6	56	7.9	56	7.6
f51m	8	8	23	5.2	23	3.8	23	4.1
misex1	8	7	14	2.5	12	2.1	12	2.1
misex2	25	18	28	2.5	28	2.9	28	3.3
misex3	14	14	169	18.8	135	50.1	135	50.2
rd73	7	3	46	15.4	22	12.5	22	12.2
rd84	8	4	107	37.9	57	22.2	56	21.1
rot	135	107	159	14.0	152	60.4	152	60.3
sao2	10	4	59	12.3	41	10.6	40	9.2
vg2	25	8	23	4.1	20	3.7	20	4.1
z4ml	7	4	4	0.2	4	0.2	4	0.2
Total			3592	817.1	2447	652.4	2441	693.0
Normalized			1	1	0.68	0.80	0.68	0.85

Algorithm 1 : R.-K. decomposition in SIS.
Algorithm 2 : R.-K. decomposition with λ set selection strategy[11].
Algorithm 3 : Algorithm 2 with compatible class encoding strategy.