BDD Based Lambda Set Selection in Roth-Karp Decomposition for LUT Architecture

Jie-Hong Jiang, Jing-Yang Jou, Juinn-Dar Huang and Jung-Shian Wei

Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan 300, Republic of China
jyjou@bestmap.ee.nctu.edu.tw

Abstract

Field Programmable Gate Arrays (FPGA's) are important devices for rapid system prototyping. Roth-Karp decomposition is one of the most popular decomposition techniques for Look-Up Table (LUT)-based FPGA technology mapping. In this paper, we propose a novel algorithm based on Binary Decision Diagrams (BDD's) for selecting good lambda set variables in Roth-Karp decomposition to minimize the number of consumed configurable logic blocks (CLB's) in FPGA's. The experimental results on a set of benchmarks show that our algorithm can produce much better results than those of the previous approach [1].

1. Introduction

The LUT-based architecture is the most popular one among many different FPGA architectures. It consists of many configurable LUT's and each LUT can implement any *k*-input functions. For example, in Xilinx XC3000 architecture [2], *k* is equal to 5.

Many technology mapping algorithms developed to minimize the number of LUT's have been successively proposed in previous studies [3-9]. Some of them first decompose the given Boolean network to be *k*-feasible. A Boolean network is said to be *k*-feasible if each node in the network has the number of its fanins to be 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, these nodes then are decomposed to be a set of *k*-feasible nodes. Although, recently there are many studies [18-21] taking advantage of BDD [11-13] for FPGA technology mapping, none of them address the relationship between BDD's and lambda set variables in Roth-Karp decomposition. In this paper, we only focus on Roth-Karp decomposition basing on BDD's in order to realize combinational circuits with the minimum number of CLB's.

We introduce some basic terms, Roth-Karp decomposition, and BDD's in Section 2. Our new algorithms will be presented in Section 3. Section 4 shows the comprehensive experimental results and comparisons with other related algorithms. The concluding remarks are given in Section 5.

2. Preliminaries

Section 2.1 introduces basic logic notations, terminology and definitions used in this paper. The definitions and principles of the classical Roth-Karp decomposition is given in Section 2.2. In Section 2.3, the BDD is introduced.

2.1 Basic Definitions and Notations

Let $B = \{0,1\}$. A **completely specified single-output function** f with an input set X consisting of n variables $x_1, x_2, ..., x_n$, is denoted as $f : B^{|X|} \rightarrow B$ where the domain $B^{|X|}$ is the Cartesian product spanned by X . A **minterm** $x = [x_1\ x_2\ ...\ x_n] \in B^{|X|}$ is a vertex in the Boolean n-space. The **on-set** of f, $X_f^{on} \subseteq B^{|X|}$, is the set of minterms x such that $f(x) = 1$, and the **off-set** of f, $X_f^{off} \subseteq B^{|X|}$, is the set of minterms x such that $f(x)=0$. A **cube** c which represents a product term (a set of minterms) p is specified by a row vector $c = [c_1\ c_2\ ...\ c_n]$ where

$c_i = 0$ if x_i appears complemented in p.
$c_i = 1$ if x_i appears not complemented in p. $1 \le i \le n$
$c_i = 2$ if x_i does not appear in p.

The **union** of two sets of cubes C and D, denoted as $C \cup D$, is a set of minterms contained by at least one cube in either C or D. A set of cubes $C = \{c^1, c^2, ..., c^k\}$ is said to be a **cover** of f if $\bigcup_{i=1}^{k} c^i$ contains all vertices of X_f^{on} and no vertex of X_f^{off}. The **matrix representation**, $f = M(C)$, of a cover is a matrix simply obtained by stacking the row vectors representing cubes contained by C. Most of the definitions can be found in [14] for more details.

2.2 Roth-Karp Decomposition

Let E be a set of variables, and let X and Y be two nonempty subsets of E such that $X \cup Y = E$ and $X \cap Y = \varnothing$[1]. Then, given a function $F : B^{|E|} \rightarrow B$, we say that $x_1, x_2 \in B^{|X|}$ are **equivalent** with respect to F, denoted as $x_1 \sim x_2$, if $\forall\ y \in B^{|Y|}$, (x_1, y) and $(x_2, y) \in B^{|E|}$ such that $F(x_1, y) = F(x_2, y)$; otherwise, x_1 is nonequivalent with x_2, denoted as $x_1 \not\sim x_2$.

Lemma 1: If F is a completely specified function, then the **transitivity** of the equivalence holds , i.e., if $x_1 \sim x_2$ and $x_2 \sim x_3$ then $x_1 \sim x_3$. ❑

Thus, all mutually equivalent elements can be grouped

[1] We only consider the disjoint Roth-Karp decomposition here.

ASP-DAC '97
0-89791-851-7$5.00

together to form an **equivalent class**, and all equivalent classes are pairwisely disjoint and the union of them is $B^{|X|}$.

Theorem 1:Given a symbolic variable *W*, and two functions $\vec{\alpha} : B^{|X|} \rightarrow W$ and $G : W \times B^{|Y|} \rightarrow B$, such that

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

holds if and only if

$$\forall x_1, x_2 \in B^{|X|}, \ \vec{\alpha}(x_1) = \vec{\alpha}(x_2) \Rightarrow x_1 \sim x_2. \quad \square$$

Equation (1) is called a **decomposition** of *F*. $\vec{\alpha}$ is a binary-input symbolic-output function. *X* is called the **bound (λ) set**, and *Y* is called the **free (μ) set**. The decomposition is disjoint if the bound set and the free set are disjoint, i.e., $X \cap Y = \varnothing$. In this paper, only the disjoint decomposition is considered.

Lemma 2: The number of elements of *W* must be no less than the number of equivalent classes in *X*. $\square$

From Lemma 2, if there are *k* equivalent classes in *X*, then $|W|$ must be no less than *k*. Let $t \geq \lceil \log_2 k \rceil$, then (1) can be rewritten as

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

where $\vec{\alpha}(x) = (\alpha_1(x), \alpha_2(x), ..., \alpha_t(x))$, $\alpha_i(x)$ is a single-output function for $1 \leq i \leq t$. It is clear that this technique can be used to reduce the number of fanins of the function under the condition $t < |X|$. Besides, the smaller *t* is, the fewer encoding functions, α_1, α_2, ... and α_t, are required. Furthermore, $2^{|t|} - |W|$ don't care can be used to simplify the function *G*. Hence, the minimum $|W|$ is desirable.

2.3 Binary Decision Diagrams

BDD's are rooted, directed acyclic graphs with two types of nodes : **terminal** and **nonterminal nodes**. The terminal nodes labeled with 0 and 1 represent the Boolean constant function 0 and 1 respectively. Each nonterminal node has two edges pointing to its children labeled **1** (or **then**) and **0** (or **else**). A nonterminal node *v* represents the Boolean function :

$$f_v(v_1,\ldots, v_i, v_{i+1},\ldots) = v_i' \cdot f_{else(v)}(v_1,\ldots, v_{i+1},\ldots) + v_i \cdot f_{then(v)}(v_1,\ldots, v_{i+1},\ldots).$$

An **ordered BDD** (**OBDD**) is a BDD with its input variables ordered and every path from the root to a terminal node in the OBDD visits the input variables in an ascending order. The order is called an **index** of the node. A **reduced OBDD** (**ROBDD**) is an OBDD of which each node represents a distinct logic function. Most of the concepts mentioned here can be found in [11-13] for more details. There is an important property between Roth-Karp decomposition and ROBDD. Let's draw a cutting line on the ROBDD such that the nonterminal nodes are partitioned into two disjoint parts. The upper part and lower part correspond to the λ set and μ set in Roth-Karp decomposition respectively. Consequently, we may observe that the nodes pointed to by the edges intersecting with the cutting line are called **equivalent class nodes**. Each equivalent class node represents one equivalent class. Therefore, we may conclude that the number of equivalent classes in Roth-Karp decomposition is the number of nodes pointed to by the edges intersecting with the cutting line in ROBDD. Another indispensable knowledge about an ROBDD is that the ordering of variables will affect the size of the ROBDD significantly. According to the previous observation, we will evolve a novel heuristic algorithm to select good λ set (or good ordering of ROBDD) such that less equivalent classes are obtained.

3. Our Approach

3.1 Variable Ordering Algorithm

Previous efforts [15, 16] of variable ordering algorithms for BDD's aim at minimizing the number of nodes in an ROBDD. These ordering methods do not aim at better Roth-Karp decomposition. In this paper, we develop a novel ordering algorithm of BDD aiming at obtaining good λ set variables. The idea is to have a variable ordering with an obvious bottleneck in the ROBDD so that there are fewer paths intersecting with the cutting line. Let's use the following example to show the basic idea of our proposed algorithm. Assume a function with its cover shown in Figure 1 and its OBDD shown in Figure 2.

If we choose variable x_1 as the first variable and x_2 as the second variable, then we have the binary tree with the x_2 and the x_2' branches of the else-edge of x_1 vanished in the resultant ROBDD. In Figure 1, the dotted circle with shading represents the else-edge of the concerned variable, x_1, and the dotted circle without shading shows the relative function information of the next ordered variable, x_2. Similarly, the solid circle with shading represents the then-edge of the concerned variable, and the solid circle without shading shows the relative function information of the next ordered variable. Because as long as input vector (x_1, x_2) has the values (0, 0) or (0, 1), all paths from the root are directed to two nodes which have the same Boolean function in the OBDD as outlined in Figure 2. Therefore, x_2 will vanish in the resultant ROBDD and some associated branches will disappear. Therefore, a good ordering should make more branches in an OBDD (binary tree) to vanish such that there will be fewer equivalent class nodes. For the Boolean function shown in Figure 1, the OBDD's are significantly different with different orderings shown in Figure 3. We can see that (x_1, x_2, x_4, x_3) is the best ordering in terms of λ set selection, if the desired number of λ set variables is two. (Note that we will regard the then-edge of node x_2 in Figure 3(a). as being deleted, because it points to a terminal node. Since there are only two terminal nodes, 0 and 1, the terminal nodes can at most add two more equivalent classes.) Let's use another Boolean function shown in Figure 4 to illustrate our ordering algorithm in more detail.

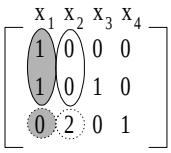


Figure 1. The cover of a Boolean function.

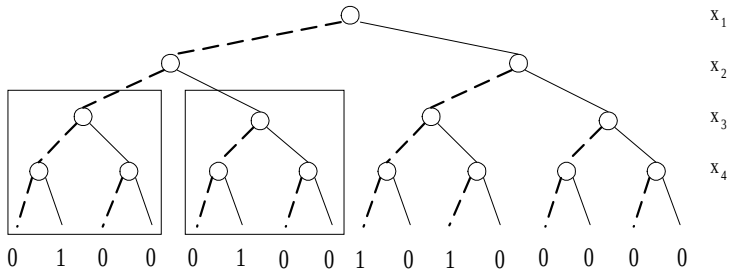
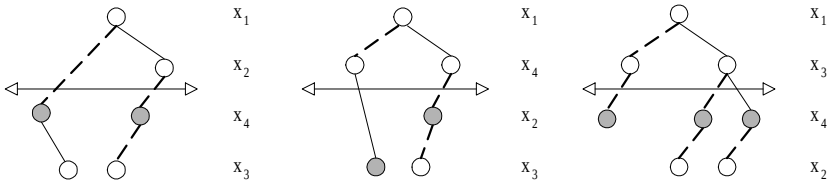


Figure 2. The OBDD of the function.



(a). (b). (c).

Figure 3. The OBDD's with different variable orderings.
(Branches pointing to terminal nodes were not shown here.)

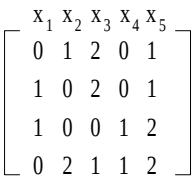


Figure 4. The cover of a Boolean function.

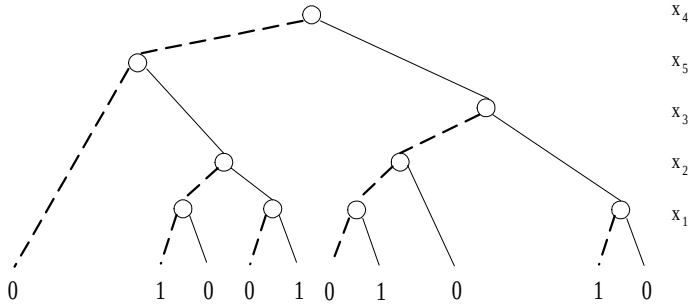


Figure 5. The partially reduced OBDD.

Step 1 : Determine the first 2 variables.

We define the benefit of each ordered pair of variables as being equal to the number of branches in the binary tree which can be deleted. For pair (x_1, x_2), branch x_2 coming from the then-edge of node x_1 in the OBDD can be deleted. This is because that every cube with x_1=1, the corresponding x_2 is always equal to 0. However, the branches of x_2 and x_2' coming from the else-edge of node x_1 can not be deleted. This is because that all cubes with x_1=0, their corresponding cubes of x_2 are not equal. There is totally one branch being deleted. Therefore, the benefit of ordered pair (x_1, x_2) is equal to 1. Similarly, for pair (x_2, x_1), the branch of x_1 coming from the then-edge of node x_2 can be deleted. The branches of x_1 and x_1' coming from the else-edge of node x_2 can not be deleted. There is totally one branch being deleted. So the benefit of the pair (x_2, x_1) is 1. By the same way, we get the benefit of each pair as following:

(x_1, x_2) : 1 (x_2, x_1) : 1 (x_3, x_1) : 0 (x_4, x_1) : 0 (x_5, x_1) : 0
(x_1, x_3) : 0 (x_2, x_3) : 0 (x_3, x_2) : 0 (x_4, x_2) : 0 (x_5, x_2) : 0
(x_1, x_4) : 0 (x_2, x_4) : 0 (x_3, x_4) : 0 (x_4, x_3) : 2 (x_5, x_3) : 0
(x_1, x_5) : 0 (x_2, x_5) : 0 (x_3, x_5) : 0 (x_4, x_5) : 3 (x_5, x_4) : 1.

Because pair (x_4, x_5) has the maximum benefit, we select them as the first two variables.

Step 2 : Select the next favorite variable.

Given the already selected variables, for each potential candidate of the next variables, evaluate how many branches can be deleted. Choose the one with the maximum benefit. If there are more than one candidate, select one randomly. In this example, after selecting (x_4, x_5) as the first two variables, for x_4' x_5 in the cover, no branch can be deleted if x_1 is selected as the next variable; no branch can be deleted if x_2 is selected as the next one; both branches x_3 and x_3' can be deleted if x_3 is the next one. For x_4 x_5' and x_4 x_5 in the cover, no branch can be deleted if x_1 is selected as the next variable; no branch can be deleted if x_2 is the next one; also no branch can be deleted if x_3 is the next one. Therefore, we can evaluate the benefits for selecting x_1, x_2, or x_3 to be the next variable as 0, 0, and 2 respectively. We thus select x_3 as the next one.

Step 3 : Iterate Step 2 until all variables are selected.

Consequently, we have the variable ordering as: (x_4, x_5, x_3, x_2, x_1). The resultant OBDD is shown in Figure 5. The

overall ordering algorithm is shown in Figure 6.

```
Our_Heuristic_Ordering(node)
{
    /* select the first two variables */
    for(all pairs of two different variables)
        var_list ← the pair which can result in deleting the most
                   branches in the OBDD;
    /* sort the rest variables */
    while(there are variables unselected){
        for(all variables unselected)
            for(all input vectors whose elements are the variables
            having been selected)
                Record how many branches can be deleted
                accumulatively;
        var_list ← the variable with the maximum benefit;
    }
    return var_list;
}
```

Figure 6. The ordering algorithm.

After we have good ordering on the BDD's, the next thing is to decide the number of λ set variables. We will discuss it in the next section.

3.2 λ Set Selection Algorithm

In the following discussion, let's assume that the target CLB can implement any 5-input functions. However, we try to decide either four, five or six variables to be the best size of λ set. The reason why we have three choice: four, five, or six, instead of only one choice, five, is because in some cases five λ set variables will cause much more equivalent classes than four or six. Given an ROBDD and the number of λ set variables temporarily selected, we can calculate the number of equivalent classes, the number of consumed CLB's and the number of remaining input variables. The costs of choosing the number of λ set variables are computed by the following rule:

cost = (# of consumed CLB's) + (# of remaining input variables) - *n*,

where *n* is the number of input variables. Since we like to have the number of remaining input variables decreased to be less than or equal to five such that we do not have to decompose it further, we give it less cost. We would like to choose the number of λ set variables with the minimum cost among four, five ,or six. If there are more than one candidate, the priority of the choices is five followed by four and followed by six. The λ set selection of our heuristic ordering algorithm is illustrated in Figure 7.

```
λ_Set_Selection_for_Our_Heuristic_Ordering(node, order)
{
    for(λ_set_size = 4, 5 or 6){
        cost[λ_set_size] ← (# of consumed CLB's) +
                           (# of remaining input variables) - n;
        #bit[λ_set_size] ← Count_Equivalent_Class(node,
                           λ_set_size,order);
        /*count the number of bits needed to encode all
        equivalent classes*/
    }
    if((#bit[4] < 4) or (#bit[5] < 5) or (#bit[6] < 6))
    /* to insure the reduction of the number of fanins of the
    function */
    return the size of λ set variables with the minimum cost;
    /*else -- size 4, 5 and 6 all failure*/
    λ_set_size ← 6;
    do{
        λ_set_size ← λ_set_size + 1;
        #bit ← Count_Equivalent_Class(node, λ_set_size,
                order);
        if((#bit < λ_set_size)
            return λ_set_size;
    }while(1);
}
```

Figure 7. The λ set selection of our heuristic ordering algorithm.

3.3 Exact λ Set Selection Algorithm

In order to obtain the best possible result, we find the optimum λ set exhaustively for the nodes whose number of input variables is less than or equal to ten. That is, we find the λ set variables from all *m* out of *n*, C(*n*, *m*), possible combinations, where *n* is the number of input variables and *m* is the number of λ set variables. We apply our heuristic ordering algorithm and λ set selection algorithm discussed in Section 3.1 and 3.2 only on the nodes of Boolean networks where the number of input variables is greater than ten.

Given the number of input variables and the number of λ set variables chosen, we can compute the number of equivalent classes based on the Boolean functions. Thus, in our exact ordering algorithm, we define the cost as the number of consumed CLB's. (If the number of λ set variables is equal to six, we have to decompose it again and will consume two times as many CLB's as the bit-num, which is the number of bits needed to encode all the equivalent classes.) If there are more than one candidate, we will choose the one with the fewest equivalent classes. The exact ordering and λ set selection algorithm is illustrated in Figure 8.

```
Exact_Ordering_and_λ_Set_Selection(node)
{
    for(λ_set_size = 4, 5 or 6)
        for(C(#fanin(node), λ_set_size) possible combinations of
        λ set variables){
            Record the minimum #bit needed,
            #min_bit[λ_set_size],
            and the relative λ set variables;
            Calculate the best cost[λ_set_size];
        }
    if((#min_bit[4] < 4) or (#min_bit[5] < 5) or (#min_bit[6] <
    6))
    /* to insure the reduction of the number of fanins of the
    function */
```

```
        return the λ set size with the minimum cost and the
        related λ set variables;
    /*else -- λ_set_size = 4, 5, and 6 all failure */
    λ_set_size ← 6;
    do{
        λ_set_size ← λ_set_size + 1;
        for(C(n, λ_set_size) possible combinations of λ set
        variables)
            Record the minimum #bit needed, #min_bit, and the
            relative λ set variables;
        if(#min_bit < λ_set_size)
        /* to insure the reduction of the number of fanins of the
        function */
            return λ_set_size and λ set variables;
    }while(1);
}
```

Figure 8. The exact ordering and λ set selection algorithm.

3.4 Proposed Roth-Karp Decomposition Algorithm

The overall decomposition algorithm is shown in Figure 9.

```
BDD_Based_RK_Decomposition(node)
{
    if(#fanin(node) <= 5)
        return;
    if(#fanin(node) <= 10)
        (order, λ_set_size) ←
        Exact_Ordering_and_λ_Set_Selection(node);
    else{
        order ← Our_Heuristic_Ordering(node);
        λ_set_size ←
        λ_Set_Selection_for_Our_Heuristic_Ordering(node);
    }
    /*random encoding*/
    (G, α functions) ← Equivalent_Class_Encoding(λ_set_size,
    order);
    for(each α function)
        BDD_Based_RK_Decomposition(α);
    BDD_Based_RK_Decomposition(G);
}
```

Figure 9. The overall decomposition algorithm.

4. Experimental Results

The algorithm described above has been integrated into SIS environment, which is developed by UC Berkeley. Experiments are conducted over a set of MCNC and ISCAS benchmark circuits to compare the results with another two implementations of Roth-Karp decomposition : one has no λ set selection strategy in mis-pga [4] as shown on column 2 of Table I, the other chooses λ set by establishing the μ-orthogonal input index table [1] as shown on column 3 of Table I. Since the structures of the initial circuits and the companioning optimization script of Roth-Karp decomposition can affect the results significantly, for the fair comparisons in these experiments, we use the most common scripts available in SIS to prepare our initial circuits. The results shown here are thus not intended for benchmarking purpose for the complete technology mapping process.

CKT	SIS	IndexTable	HEU	HEU	HEU+ EXACT	HEU+ EXACT
	RK_dec	RK_dec	RK_dec	CPU time	RK_dec	CPU time
name	nodes	nodes	nodes	sec	nodes	sec
5xp1	21	19	26	1.6	19	2.3
9sym	7	6	9	5.2	6	22.2
alu2	122	77	102	25.1	70	98.8
alu4	381	239	221	8.0	216	36.6
apex4[2]	984	426	439	26.1	354	214.8
apex6	229	232	211	4.9	209	52.5
apex7	63	65	62	2.7	63	3.1
b9	38	37	36	1.8	36	1.6
clip	41	32	30	27.5	23	42.9
count	31	31	31	1.1	31	1.0
des	1919	1211	1019	73.1	991	379.7
duke2	177	125	125	8.0	117	26.5
e64	80	80	80	2.2	80	2.2
f51m	23	16	22	2.4	15	3.7
misex1	17	16	16	1.0	17	1.6
misex2	32	32	32	1.1	32	3.2
misex3	223	165	166	13.9	154	28.8
rd73	8	8	8	2.4	8	14.9
rd84	13	13	16	6.9	13	40.4
rot	225	195	188	31.7	191	25.2
sao2	52	37	46	6.1	27	36.4
vg2	28	23	24	0.8	23	4.0
z4ml	5	5	5	0.5	5	0.9
C499	70	70	70	4.6	70	8.8
C880	170	106	89	14.4	87	17.7
Total	4959	3266	3073	273.1	2857	1069.8
Normalize1	1	0.659	0.620		0.576	
Normalize2	1.518	1	0.941		0.875	

Table I : The experimental results of benchmark circuits.

Among multi-level logic synthesis techniques, the **collapse** operation, which collapses a multi-level network into a two-level network, can have dramatic impacts on the quality of outputs. However, applying the collapse operation on some circuits can take forever and consume a lot of memory space without producing the final outputs. In this experiment, we adopt the strategy from mis-pga to prepare two scripts.

Script C contains the following SIS commands

collapse

[2] Multi-level network could not finish. We only report the time consumed in the two-level circuit.

simplify -d.

Script S is the SIS standard optimization script.

To prepare the initial circuits, we apply both scripts on the circuits with at most 10 primary inputs and report the best results. For circuits with more than 10 primary inputs, we only apply script S. After obtaining the initial networks, the same mapping script

```
xl_k_decomp -n 5    /*Three Roth-Karp decomposition
                      algorithms are applied here*/
xl_partition -n 5 -tm
xl_cover
```

is used in all experiments to obtain final networks. Thus, each node in these networks may be one-to-one mapped into a 5-input CLB. The results are shown in Table I.

On columns 4 and 5 of Table I, we also report the results on only using the heuristic ordering algorithm. We can see that the heuristics alone does not do as good as our heuristics plus exact algorithm (columns 6 and 7) even though the CPU time consumption is less. This is the reason why we use the number of fanins of nodes as the criteria to switch between the heuristic and the exact algorithm. On average, our algorithm produces 42.4% fewer nodes than that of SIS while the Index-Table algorithm [1] produces 34.1% fewer nodes than that of SIS. As we compare our approach with the Index-Table algorithm, it produces 12.5% fewer nodes than that of the Index-Table algorithm. Indeed, our algorithm produces the best results for all the benchmarking circuits except misex1. Since the CPU time available from other two approaches are based on SUN SPARC 2, and the CPU time reported by all three algorithms are reasonably small and is of no concerned, therefore, in the table, we only report the CPU time of our algorithm running on SUN SPARC 20 workstation.

5. Conclusions

In this paper, we propose a novel heuristic algorithm to select λ set variables in the Roth-Karp decomposition for better LUT utilization. The result shows that our algorithm performs much better than the existing methods.

References

[1] Wen-Zen Shen, Juinn-Dar Huang, and Shin-Min Chao, "Lambda Set Selection in Roth-Karp Decomposition for LUT-Based FPGA Technology Mapping," Proceedings 32nd Design Automation Conf., pp.65-69, June 1995.

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

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

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

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

[6] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf : Fast Technology Mapping for Lookup Table-Based FPGA's," Proceedings 28th Design Automation Conf., June 1991.

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

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

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

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

[11] C. Y. Lee, " Representation of Switching Circuits by Binary-Decision Programs," Bell System Technical J., vol. 38, pp.985-999, July 1959.

[12] S. B. Akers, " Binary Decision Diagrams," IEEE Trans. on Computers, vol. C-27, pp.509- 516, June 1978.

[13] Randal E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans. on Computers, vol. C-35, pp.677-691, August 1986.

[14] R. K. Brayton, C. McMullen, G. D. Hachtel, and A. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis," Kluwer Academic Publishers, 1984.

[15] Steven J. Friedman and Kenneth J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," IEEE Trans. on Computers, vol. 39, pp.710-713, May 1990.

[16] Masahiro Fujita, Hisanori Fujisawa, and Yusuke Matsunaga, "Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evaluation," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 12, pp.6-12, January 1993.

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

[18] Yung-Te Lai, Massound Pedram and Sarma B.K. Vrudhula, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," Proceedings 30th Design Automation Conf., pp.642-647, June 1993.

[19] Bernd Wurth, Klaus Eckl, and Kurt Antreich, "Functional Multiple-Output Decomposition: Theory and an Implicit Algorithm," Proceedings 32th Design Automation Conf., pp.54-59, June 1995.

[20] Hiroshi Sawada, Takayuki Suyama and Akira Nagoya, "Logic Synthesis for Look-Up Table based FPGAs using Functional Decomposition and Support Minimization," Proceedings Int. Conf. Computer-Aided Design, pp. 353-358, Nov. 1995.

[21] Christoph Scholl and Paul Molitor, "Communication Based FPGA Synthesis for Multi-Output Boolean Functions," Proceedings Asia and South Pacific Design Automation Conf., pp.279-287, 1995.