Lambda Set Selection in Roth-Karp Decomposition for LUT-Based FPGA Technology Mapping

Wen-Zen Shen, Juinn-Dar Huang, and Shih-Min Chao

Department of Electronics Engineering, National Chiao Tung University,
Hsinchu, Taiwan, the Republic of China

Abstract

Roth-Karp decomposition is a classical decomposition method. Because it can reduce the number of input variables of a function, it becomes one of the most popular techniques used in LUT-based FPGA technology mapping. However, the lambda set selection problem, which can dramatically affect the decomposition quality in Roth-Karp decomposition, has not been formally addressed before. In this paper, we propose a new heuristic-based algorithm to solve this problem. The experimental results show that our algorithm can efficiently produce outputs with better decomposition quality than that produced by other algorithms without using lambda set selection strategy.

1. Introduction

Field Programmable Gate Arrays (FPGAs) 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 very popular to be used 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 identical logic blocks which can implement any Boolean function with k inputs. For example, in Xilinx XC3000 architecture[1], k is equal to 5. Many tools developed for LUT-based FPGA technology mapping have been proposed in previous studies[2-8]. Most of them 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 fanin is no more than k. Hence, the corresponding circuit can be directly realized by a one-to-one mapping between nodes and LUTs. If there are some nodes that are not k-feasible, then they have to be decomposed to a set of k-feasible nodes. Many decomposition techniques, such as AND-OR decomposition, cofactoring, disjoint decomposition, if-then-else DAG, communication complexity reduction[9] and Roth-Karp decomposition, are widely used in logic synthesis. In this paper, we concentrate on Roth-Karp decomposition[10]. This method can decompose a Boolean function with a given input partition. In general, the quality of this decomposition strongly depends on what the input partition is. Some previous studies showed that grouping symmetric input variables into the same partition tends to produce better results. However, to the best of our knowledge, finding a good input partition in Roth-Karp decomposition has not been formally addressed in previous research. In this paper, we propose a new heuristics to solve this problem.

This paper is organized as follows. Section 2 briefly introduces some terminologies used in this paper. Section 3 describes Roth-Karp decomposition and a classical implementation for it. In Section 4, our newly developed heuristics for selecting a good λ set is given in detail. Section 5 shows some experimental results and concluding remarks are given in Section 6.

32nd ACM/IEEE Design Automation Conference ®

2. Preliminaries

Some basic notations, terminologies and definitions used in this paper are given in this section and most of them can be found in [11] for more details.

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

$$c_i = \begin{cases} 0 & \text{if } x_i \text{ appears complemented in p.} \\ 1 & \text{if } x_i \text{ appears not complemented in p.} \quad 1 \le i \le n. \\ 2 & \text{if } x_i \text{ does not appear in p.} \end{cases}$$

The **size** of a cube c, denoted as $\pi(c)$, indicates the number of minterms contained by c. The cube d is said to be **contained** by the cube c, denoted as $c \supseteq d$, if all minterms contained by d is also contained by c. The **intersection** of two sets of cubes C and D, denoted as $C \cap D$, is also a set of minterms (possibly empty) contained both by C and D. Two cubes c and d are said to be **orthogonal** if $c \cap d = \varnothing$. 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 $C = \{ c^1, c^2, \ldots, c^k\}$ 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 C is a matrix simply obtained by stacking the row vectors representing cubes contained by C.

3. Classical Roth-Karp Decomposition

In this section, a brief introduction to Roth-Karp decomposition and a classical implementation method are both given.

Let E and W be two sets of variables, and let X and Y be two nonempty subsets of E such that $X \cup Y = E$. Then, given a function $F : E \to B$, we say that $x_1, x_2 \in X$ are **compatible** with respect to F, denoted as $x_1 \sim x_2$, if $\forall\, y \in Y$, (x_1, y) and $(x_2, y) \in E$ such that $F(x_1, y) = F(x_2, y)$; otherwise, x_1 is incompatible with x_2, denoted as $x_1 \,!\!\sim x_2$.

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

Thus, all mutually compatible elements can be grouped together to form a **compatible class**, and all compatible classes are pairwisely disjoint.

Theorem: Given two functions $\vec{\alpha} : X \to W$ and $G : W{\times}Y \to B$, such that

$$\forall\,(x, y) \in E,\ F(x, y) = G(\vec{\alpha}(x), y) \qquad (1)$$

holds if and only if

$$\forall\, x_1, x_2 \in X,\ \vec{\alpha}(x_1) = \vec{\alpha}(x_2) \Rightarrow x_1 \sim x_2.$$

Equation (1) is called a **decomposition** of F. $\vec{\alpha}$ could be a multiple-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. The **λ cube** of a cube c, denoted as c_λ, is a subvector of c which contains entries corresponding to the variables in the λ set. The **μ cube** of c, denoted as c_μ, is defined similarly. A **λ minterm** $m_\lambda \in B^{|\lambda|}$ is a λ cube which does not contain any 2s. The **μ cover** of a cover C, denoted as C_μ, is the set containing all μ cubes in C.

Lemma 2: The number of elements of W must be no less than the number of compatible classes in X.

From Lemma 2, if there are k compatible classes in X, then |W| must be no less than $\lceil \log_2 k \rceil$. 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 $t < |X|$.

Lemma 3: Given C and D are covers of F and $\overline{F}$, respectively, if there exists a cube c in C and a cube d in D such that $c_\mu \cap d_\mu \neq \varnothing$, then for all λ minterms $m_{\lambda 0}$ contained by c_λ and all λ minterms $m_{\lambda 1}$ contained by d_λ, $m_{\lambda 0} \,!\!\sim m_{\lambda 1}$.

Thus, the compatible classes can be identified by applying Lemma 1 and 3. Firstly, a complete graph G(V, E) is constructed where $|V| = 2^{|\lambda|}$. Each vertex represents a λ minterm and an edge represents the compatibility of two connected vertices (λ minterms) in G. Secondly, all edges (v_0, v_1) can be deleted where λ minterms represented by v_0 and v_1 are found to be incompatible by Lemma 3. Finally, from Lemma 1, each connected subgraph is a complete graph, and hence represents a compatible class. Finding all connected subgraphs is a very simple work, so compatible classes are also easily identified. Notice that the number of connected subgraphs is equal to the number of compatible classes.

Example 1:

Given $X = \{x_1, x_2, x_3, x_4\}$, $\lambda = \{x_1, x_2\}$, $\mu = \{x_3, x_4\}$,

$$F = M(C) = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 1 \\ 1 & 1 & 0 & 2 \\ 1 & 1 & 2 & 1 \end{bmatrix}, \text{ and } \overline{F} = M(D) = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 2 & 0 \end{bmatrix},$$

identify all compatible classes.

According to Lemma 3, we obtain that

$$\because\ c_\mu^2 \cap d_\mu^3 \neq \varnothing \quad \Rightarrow \quad [0\ 0] \,!\!\sim [0\ 1]$$

$$\because\ c_\mu^2 \cap d_\mu^5 \neq \varnothing \quad \Rightarrow \quad [0\ 0] \,!\!\sim [1\ 0]$$

$$\because\ c_\mu^4 \cap d_\mu^3 \neq \varnothing \quad \Rightarrow \quad [1\ 1] \,!\!\sim [0\ 1]$$

$$\because\ c_\mu^4 \cap d_\mu^5 \neq \varnothing \quad \Rightarrow \quad [1\ 1] \,!\!\sim [1\ 0]$$

The corresponding compatibility graph is illustrated in Fig. 1.

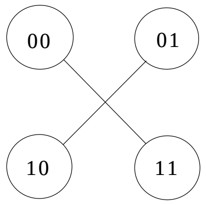

Fig. 1 : Compatibility graph in Example 1.

It is obvious that two compatible classes, {[0 0], [1 1]} and {[0 1], [1 0]}, are identified.

When applying Roth-Karp decomposition for LUT-based FPGA technology mapping, the major problem is how to choose a good λ set instead of how to decompose a node with a given one. Unfortunately, no useful information about selecting a good λ set can be extracted from the implementation described above. In the next section, another implementation of Roth-Karp decomposition will be given and a novel heuristics on picking a good λ set are proposed.

4. Lambda Set Selection

As shown below, Lemma 3 can be simply rewritten as Lemma 4.

Lemma 4: Given $C = \{c^1, c^2, \ldots, c^i\}$ and $D = \{d^1, d^2, \ldots, d^j\}$ are covers of F and $\overline{F}$, respectively; two λ minterms, $m_{\lambda 0}$ and $m_{\lambda 1}$, are said to be compatible if and only if

$$C' = \{c \mid c^k_\lambda \supseteq m_{\lambda 0},\ 1 \le k \le i\},$$

$$D' = \{d \mid d^k_\lambda \supseteq m_{\lambda 1},\ 1 \le k \le j\} \Rightarrow C'_\mu \cap D'_\mu = \varnothing, \text{ and}$$

$$C'' = \{c \mid c^k_\lambda \supseteq m_{\lambda 1},\ 1 \le k \le i\},$$

$$D'' = \{d \mid d^k_\lambda \supseteq m_{\lambda 0},\ 1 \le k \le j\} \Rightarrow C''_\mu \cap D''_\mu = \varnothing \text{ hold.}$$

In order to apply Lemma 4, an $i \times j$ **μ-orthogonal matrix** O is created such that $O_{mn} = 1$ if c^m_μ is orthogonal to d^n_μ, otherwise $O_{mn} = 0$, for $1 \le m \le i$ and $1 \le n \le j$. A $2 \times 2^{|\lambda|}$ **cube index table** I is also constructed such that I_{1n} is a set containing all cube indices i where $c^i_\lambda \supseteq$ the n^{th} λ minterm and I_{2n} is a set containing all cube indices i where $d^i_\lambda \supseteq$ the n^{th} λ minterm. Then, Lemma 4 can be transformed into Lemma 5.

Lemma 5: The i^{th} λ minterm and the j^{th} λ minterm are said to be compatible if and only if

O', a submatrix of O, contains rows specified by I_{1i} and columns specified by I_{2j}, and O'', a submatrix of O, contains rows specified by I_{1j} and columns specified by I_{2i} such that all elements in O' and O'' are 1s.

Example 2:

Check whether [0 0] and [0 1] are compatible in Example 1.

The corresponding μ-orthogonal matrix and the cube index table are shown in Fig. 2(a) and Fig. 2(b), respectively.

$$O = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Fig. 2(a) : The μ-orthogonal matrix.

λ minterm	[0 0]	[0 1]	[1 0]	[1 1]
cube index of on-set	{1, 2, 3}	{1}	{1}	{1, 4, 5}
cube index of off-set	{1}	{1, 2, 3}	{1, 4, 5}	{1}

Fig. 2(b) : The cube index table.

According to Lemma 5, we obtain

$$O' = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \text{, and} \quad O'' = [1]$$

Thus, [0 0] and [0 1] are found to be incompatible.

From the implementation described above, any 0 in either O' or O'' will make two corresponding λ minterms to be incompatible. So it is intuitive to predict that fewer compatible classes will be found if there are fewer 0s in the μ-orthogonal matrix. Based on this idea, we propose a novel heuristics for λ set selection in the following discussion.

Assume that $C = \{c^1, c^2, \ldots, c^i\}$ and $D = \{d^1, d^2, \ldots, d^j\}$ are covers of F and $\overline{F}$, respectively. An $i \times j$ **μ-orthogonal input index table** R is constructed such that R_{mn} is a set of input indices k where $x_k \in \mu$ and $c^m_k \cap d^n_k = \varnothing$. Notice that $O_{mn} = 0$ only when R_{mn} is $\varnothing$. At first, all input variables are put into μ set and thus no entry in R is $\varnothing$. But after picking some variables out from the μ set to form the λ set, some entries in R might become empty sets and it means that some elements in O become 0s. The heuristics that we propose here is to define a reasonable weight function to pick input variables out such that the reference count of 0s in O' and O'' is minimized while checking the compatibility by Lemma 5. The weight function W of an input variable x_k is defined as

$$W(x_k) = \sum_{m=1}^{i} \varpi\left(x_k, c^m\right) + \sum_{m=1}^{j} \varpi\left(x_k, d^m\right) \qquad (2)$$

where

$$\varpi\left(x_k, c^m\right) =$$

$$\pi(c^m_\lambda) * \text{old\_\#0}(m) \qquad \text{if } c^m_k = 2,$$

$$\pi(c^m_\lambda) * \sum_{n=1}^{j} \text{inc\_\#0}\left(m, n, x_k\right) \qquad \text{if } c^m_k = 0 \text{ or } 1.$$

$$\varpi\left(x_k, d^m\right) =$$

$$\pi(d^m_\lambda) * \text{old\_\#0}(m) \qquad \text{if } d^m_k = 2,$$

$$\pi(d^m_\lambda) * \sum_{n=1}^{i} \text{inc\_\#0}\left(n, m, x_k\right) \qquad \text{if } d^m_k = 0 \text{ or } 1.$$

While calculating $\varpi\left(x_k, c^m\right)$ in (2), $\pi(c^m_\lambda)$ estimates how many times the m^{th} row of O tends to be referenced while checking the compatibility and is initialized to 1. old_#0(m) represents how many 0s in the m^{th} row of O currently and should be initialized to 0. When c^m_k = 2, it never generates any new 0 in O but just doubles the size of c^m_λ. Hence, if x_k is out of the μ set, the product of $\pi(c^m_\lambda)$ and old_#0(m) gives an estimation about the newly increased reference count of 0s while checking the compatibility. When c^m_k = 0 or 1, inc_#0(m, n, x_k) estimates a potential of a 0 being newly generated in the m^{th} row of O if x_k is out of the μ set. inc_#0(m, n, x_k) is defined as

$$\text{inc\_\#0}(m, n, x_k) = \begin{cases} 0 & \text{if } R_{mn} = \varnothing \text{ or } k \notin R_{mn} \text{ or } |R_{mn}| > \text{the number of variables still to be selected,} \\ 2^{1 - |R_{mn}|} & \text{otherwise.} \end{cases}$$

Since c^m_k = 1 or 0, it might increase the number of 0s in O but never affects the size of c^m_λ. Then the product of $\pi(c^m_\lambda)$ and $\sum_{n=1}^{j} \text{inc\_\#0}\left(m, n, x_k\right)$ can also estimate about newly increased reference counts of 0s while checking the compatibility. $\varpi\left(x_k, d^m\right)$ is defined similarly. In our algorithm, the weights of all candidates are calculated. The input variable with the minimum weight is selected to join the λ set and then the related information is updated. This procedure does not terminate until the desired λ set size is achieved.

Example 3:
Given a function F defined in Example 1, chose a λ set with two input variables such that the number of compatible classes is minimum.

At the beginning, the λ set is empty and then the μ-orthogonal input index table is shown below

ON\OFF	1	2	3	4	5
1	{3, 4}	{3}	{4}	{3}	{4}
2	{3}	{2, 3}	{2}	{1, 3}	{1}
3	{4}	{2}	{2, 4}	{1}	{1, 4}
4	{3}	{1, 3}	{1}	{2, 3}	{2}
5	{4}	{1}	{1, 4}	{2}	{2, 4}

The weight functions are calculated according to (2).

W(x_1) = (1*0 + 1*3/2 + 1*3/2 + 1*3/2 + 1*3/2) + (1*0 + 1*3/2 + 1*3/2 + 1*3/2 + 1*3/2) = 12
W(x_2) = (1*0 + 1*3/2 + 1*3/2 + 1*3/2 + 1*3/2) + (1*0 + 1*3/2 + 1*3/2 + 1*3/2 + 1*3/2) = 12
W(x_3) = (1*5/2 + 1*2 + 1*0 + 1*2 + 1*0) + (1*5/2 + 1*2 + 1*0 + 1*2 + 1*0) = 13
W(x_4) = (1*5/2 + 1*0 + 1*2 + 1*0 + 1*2) + (1*5/2 + 1*0 + 1*2 + 1*0 + 1*2) = 13

Since W(x_1) and W(x_2) are both minimum, x_1 is chosen randomly. Then, the μ-orthogonal input index table becomes

ON\OFF	1	2	3	4	5
1	{3, 4}	{3}	{4}	{3}	{4}
2	{3}	{2, 3}	{2}	{3}	∅
3	{4}	{2}	{2, 4}	∅	{4}
4	{3}	{3}	∅	{2, 3}	{2}
5	{4}	∅	{4}	{2}	{2, 4}

The weight functions are

W(x_2) = (2*0 + 1*3/2 + 1*3/2 + 1*3/2 + 1*3/2) + (2*0 + 1*3/2 + 1*3/2 + 1*3/2 + 1*3/2) = 12
W(x_3) = (2*5/2 + 1*5/2 + 1*0 + 1*5/2 + 1*0) + (2*5/2 + 1*5/2 + 1*0 + 1*5/2 + 1*0) = 20
W(x_4) = (2*5/2 + 1*5/2 + 1*0 + 1*5/2 + 1*0) + (2*5/2 + 1*5/2 + 1*0 + 1*5/2 + 1*0) = 20

Obviously, x_2 should be selected. Thus, the λ set is {x_1, x_2}. This is the best selection since it results in the minimum number of compatible classes in this example.

5. Experimental Results

The algorithm described above has been integrated into SIS environment which is developed by UC Berkeley and is a superset of mis-pga[3, 12]. In order to demonstrate the quality and the efficiency of our algorithm, two experiments are conducted over a large set of MCNC and ISCAS benchmark circuits to compare with another implementation, which has no λ set selection strategy, of Roth-Karp decomposition in mis-pga. One experiment is performed on two-level circuits obtained by the SIS script

```
collapse
simplify -d
```

and the other is performed on multi-level circuits obtained by the SIS standard optimization script. After obtaining the initial networks, the same mapping script

```
xl_k_decomp  -n 5
xl_partition   -n 5 -tm
xl_cover
```

is used in both experiments to obtain 5-feasible networks. Thus, each node in these networks can be one-to-one mapped into a 5-input LUT directly. The experiments are run on a SUN SPARC 2 workstation and the results are shown in Table I and II, respectively.

From Table I, we can see that out of 16 two-level circuits, OUR_RK_decomp is better on 13 cases and even on the rest of 3 cases when compared with SIS_RK_decomp. On average, our algorithm produces 27% fewer nodes than that of SIS. Moreover, this algorithm is also very time-efficient. The CPU time taken by SIS is 6.3 times more than that used by our algorithm to complete this experiment. From Table II, out of 26 multi-level circuits, OUR_RK_decomp is better on 16 cases, worse on 3 cases and even on the rest of 7 cases. On average, our algorithm produces 29% fewer nodes than that of SIS, and also takes 21% less CPU time than SIS in this experiment.

6. Conclusions

In this paper, we propose a novel heuristics to select a good λ set in Roth-Karp decomposition. Experimental results show that our algorithm can efficiently use fewer extra nodes to decompose either a two-level network or a multi-level network to satisfy the given fanin constraint than a plain implementation of Roth-Karp decomposition. Thus, this new technique is especially useful in the field of combinational circuit synthesis for LUT-based FPGA technology mapping.

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] TingTing Hwang, Robert M. Owens, and Mary J. Irwin, "Efficiently Computing Communication Complexity for Multilevel Logic Synthesis," in *IEEE Trans. on Computer-Aided Design*, May 1992, pp.545-554.

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

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

[12] 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 benchmark circuits.

	SIS_RK_decomp		OUR_RK_decomp	
CKT name	#nodes	time(sec)	#nodes	time(sec)
5xp1	21	5.2	19	3.1
9sym	7	4.8	6	8.2
alu2	122	182.7	77	51.4
apex4	984	3418.6	426	243.4
b9	115	55.6	92	46.5
clip	79	50.9	36	21.6
count	80	54.9	52	70.2
duke2	763	2870.8	722	450.8
e64	544	73.6	544	117.2
f51m	23	3.6	16	3.3
misex1	17	2.2	16	3.8
misex2	49	12.6	43	10.8
rd73	8	3.3	8	6.9
rd84	13	8.4	13	23.2
sao2	52	19.6	37	16.0
z4ml	10	3.1	6	1.6
Total	2887	6769.9	2113	1078.0

Tabel II: The experimental results of multi-level benchmark circuits.

	SIS_RK_decomp		OUR_RK_decomp	
CKT name	#nodes	time(sec)	#nodes	time(sec)
5xp1	26	6.7	26	4.9
9sym	111	48.3	74	40.6
9symml	96	50.3	76	32.0
bw	49	35.4	56	27.8
duke2	177	111.3	125	55.9
f51m	27	12.8	27	9.1
misex1	19	3.5	18	2.7
misex2	32	6.5	32	6.5
misex3	223	22.7	165	87.4
rd53	3	5.8	3	2.2
rd73	55	28.2	30	29.8
rd84	129	72.7	75	35.2
sao2	72	26.2	51	20.2
vg2	28	8.1	23	7.4
z4ml	5	0.4	5	0.4
alu2	190	140.9	119	97.8
alu4	381	83.6	239	52.8
apex6	229	21.5	232	27.3
apex7	63	15.2	65	17.3
b9	38	8.6	37	8.9
count	31	5.7	31	5.7
des	1919	624.7	1211	405.0
rot	225	24.4	195	116.8
clip	41	16.6	32	13.4
e64	80	14.8	80	14.6
C499	70	27.4	70	27.8
C880	170	83.9	106	41.0
Total	4489	1506.2	3203	1190.5