Restructuring Logic Representations with Easily Detectable Simple Disjunctive Decompositions

Hiroshi Sawada, Shigeru Yamashita and Akira Nagoya
NTT Communication Science Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, JAPAN

Abstract

Simple disjunctive decomposition is a special case of logic function decompositions, where variables are divided into two disjoint sets and there is only one newly introduced variable. This paper presents that many simple disjunctive decompositions can be found easily by detecting symmetric variables or checking variable cofactors. We also propose an algorithm that constructs a new logic representation for a simple disjunctive decomposition by assigning constant values to variables in the original representation. The algorithm enables us to apply the decomposition with keeping good structures of the original representation. We have performed experiments to restructure fanout free cones of multi-level logic circuits, and obtained better results than when not restructuring them.

1. Introduction

Decomposition of a logic representation is an important operation in multi-level logic synthesis. A simple disjunctive decomposition $f(X,Y) = h(g(X),Y)$ is a special case of logic decompositions, where variables of f are divided into two disjoint sets, X and Y, and g is a single-output function [1, 2, 3]. If a function has a simple disjunctive decomposition, we had better apply the decomposition to obtain a good multi-level logic representation.

To examine whether a simple disjunctive decomposition exists for f and X, we have to calculate all resultant functions assigning constant values $\{0,1\}^{|X|}$ to X and check whether the number of distinct functions is only two. Recently, efficient methods [4, 5, 6] using an ordered binary decision diagram (OBDD) [7] have been proposed. In these methods, all distinct functions are constructed in an OBDD with a variable order where X precedes Y. It is, however, difficult to find an appropriate variable set X that gives a decomposition. The number of possible variable sets grows exponentially with the number of variables.

In [8], we have shown that a set of symmetric variables is a good candidate for X that gives a decomposition. Thanks to the idea of asymmetry [9], detecting symmetric variables is not an expensive operation. Thus, such decompositions are easily detectable. In this paper, we show that simple disjunctive decompositions using a 2-input function at the output are also easily detectable. They can be detected by checking variable cofactors of the function, and this task is performed by traversing OBDD nodes.

For cases of simple disjunctive decompositions, it is not straightforward to construct the representations of new functions g and h, because decomposition test is performed on the logic function f of an original representation and we can obtain only functions for g and h. Constructing new representations from scratch ignores good structures (for example, optimum sum-of-products forms or multi-level forms) in the original representation, and moreover is time consuming. In this paper, we propose an algorithm that constructs new logic representations by assigning constant values to variables in the original representation. By using the algorithm, we can apply simple disjunctive decompositions with keeping good structures in the original representation.

This paper is organized as follows. In Section 2, we give some definitions for simple disjunctive decompositions and symmetric variables. Section 3 shows our method to detect simple disjunctive decompositions of a function. In Section 4, we show an algorithm that constructs new logic representations when a simple disjunctive decomposition exists. Section 5 presents experimental results. We conclude this paper in Section 6.

2. Preliminaries

2.1. Simple disjunctive decomposition

A simple disjunctive decomposition of a function $f(X,Y)$: $\{0,1\}^{|X|+|Y|} \rightarrow \{0,1\}$ is of the form

$$f(X,Y) = h(g(X),Y),$$

where X and Y are sets of variables such that $X \cap Y = \emptyset$, and g: $\{0,1\}^{|X|} \rightarrow \{0,1\}$ and h: $\{0,1\}^{|Y|+1} \rightarrow \{0,1\}$ are completely specified logic functions. The sets X and Y

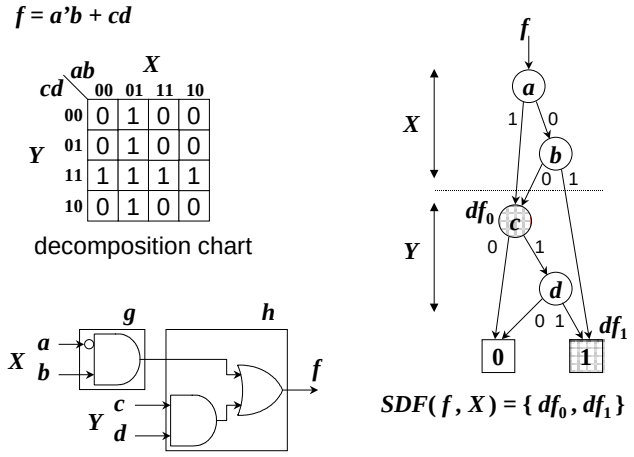

Figure 1. A simple disjunctive decomposition

are called the **bound set** and the **free set**. In this paper, g is called the **subfunction** and h is called the **image**.

Ashenhurst [1] used a decomposition chart to examine whether there exists a simple disjunctive decomposition. It is a truth table where the columns correspond to the bound set and the rows correspond to the free set. If the number of distinct column vectors is two, there exists a simple disjunctive decomposition. In this paper, we define a **set of distinct functions** $SDF(f, X)$ for a function f and a bound set X as follows; $SDF(f, X) = \{df \mid df = f(\epsilon, Y), \forall \epsilon \in \{0,1\}^{|X|}\}$. It corresponds to a set of distinct column vectors in a decomposition chart. Recently, several researchers [4, 5, 6] have proposed a method to calculate $SDF(f, X)$ efficiently by constructing an OBDD of f where variables in X are ordered before variables in Y.

When a simple disjunctive decomposition exists for f, it can be expressed as $f = g(X)' \cdot df_0(Y) + g(X) \cdot df_1(Y)$, where $SDF(f, X) = \{df_0(Y), df_1(Y)\}$. Therefore, g is obtained by replacing $df_0(Y)$ and $df_1(Y)$ with 0 and 1, and h is given by $h = g' \cdot df_0(Y) + g \cdot df_1(Y)$.

Figure 1 shows an example. For a function $f = a'b + cd$, a simple disjunctive decomposition exists for a bound set $\{a, b\}$ because $SDF(f, \{a, b\}) = \{cd, 1\}$ and the size is two. The subfunction g is obtained by replacing df_0 and df_1 with 0 and 1. Thus $g = a'b$. The image is $h = g' \cdot cd + g \cdot 1 = cd + g$.

2.2. Symmetric variables

The **cofactors** of f with respect to $x_i = 1$ and $x_i = 0$ are $f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ and $f_{x_i'} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$, respectively.

A function $f(x_1, \ldots, x_n)$ is **symmetric** in $\{x_i, x_j\}$ (or $\{x_i, x_j'\}$) if the interchange of x_i and x_j (or x_j') leaves the function invariant. A function f is symmetric in $\{x_i, x_j\}$ (or $\{x_i, x_j'\}$) if and only if $f_{x_i x_j'} = f_{x_i' x_j}$ (or $f_{x_i x_j} = f_{x_i' x_j'}$).

For a complete specified function, symmetry is an equivalence relation. Thus, symmetry in a set of variables can be calculated from a set of symmetries in two variables. We call the set of variables a **symmetric variable group**. For example, a function $f = ab'c + d$ is symmetric in $\{a, b'\}$ and $\{b', c\}$, and therefore $\{a, b', c\}$ is a symmetric variable group.

In addition, we also use the following definitions found in [10]. A function f is **multiform symmetric** in variables x_i and x_j if f is symmetric in both $\{x_i, x_j\}$ and $\{x_i, x_j'\}$. A function f is **single-variable symmetric** in x_i in the space $x_j = 1$ if x_i's two cofactors of f_{x_j} are identical; $f_{x_i' x_j} = f_{x_i x_j}$. A function f is single-variable symmetric in x_i in the space $x_j = 0$ if x_i's two cofactors of $f_{x_j'}$ are identical; $f_{x_i' x_j'} = f_{x_i x_j'}$.

If f has a symmetric variable group and f is multiform symmetric (or single-variable symmetric) in a pair of variables in the group, then f is multiform symmetric (or single-variable symmetric) in all pairs of variables in the group.

3. Easily detectable simple disjunctive decompositions

This section shows that many simple disjunctive decompositions can be found easily by detecting symmetric variables or checking variable cofactors.

3.1. By detecting symmetric variables

In [8], we have shown that a symmetric variable group is a good candidate for the bound set that gives a decomposition. Here, we summarize the decomposition procedure.

1. Detect the symmetric variable groups of a function f. Symmetric variables can be detected at low computation cost by applying the idea of asymmetry [9] to filter out the possibility of symmetries, which is examined by depth-first traversals in an OBDD. As for pairs of variables that are not filtered out by the asymmetry check, we perform depth-first traversals in the OBDD again to examine symmetries [8].

2. If a symmetric variable group S contains multiform symmetric variables, $f_{ab} = f_{a'b'}$ and $f_{ab'} = f_{a'b}$ hold for each variable pair $a, b \in S$. This leads to the size of $SDF(f, S)$ is two, and there exists a simple disjunctive decomposition of f whose bound set is S. The subfunction is obtained by replacing the two elements of $SDF(f, S)$ with 0 and 1. It is implemented in an XOR gate.

3. If a symmetric variable group S contains single-variable symmetric variables, we can invert some of

the variables in S such that $f_{a'b'} = f_{ab'} = f_{a'b}$ holds for each variable pair $a, b \in S$. This leads to the size of $SDF(f, S)$ is two, and there exists a simple disjunctive decomposition of f whose bound set is S. The subfunction is obtained by replacing the two elements of $SDF(f, S)$ with 0 and 1. It is implemented in an AND gate and some inverters.

4. For a symmetric variable group that contains neither multiform symmetric variables nor single-variable symmetric variables, calculate $SDF(f, S)$ for the group S and examine whether a simple disjunctive decomposition exists. If a decomposition exists, the subfunction is a symmetric function.

The computation cost of the above procedure is not expensive because candidate bound sets are just symmetric variable groups. We can say that decompositions whose subfunction is a symmetric function are easily detectable.

3.2. By checking variable cofactors

Here, we show another type of simple disjunctive decompositions, which can be detected by checking variable cofactors. They are ones whose image is a 2-input function. There are five cases for such decompositions. We summerize the forms and the conditions for existence.

- If $f_{x'} = (f_x)'$, f has a decomposition $f = x \oplus f_{x'}$.
- If $f_{x'} = 0$, f has a decomposition $f = x \cdot f_x$.
- If $f_x = 0$, f has a decomposition $f = x' \cdot f_{x'}$.
- If $f_{x'} = 1$, f has a decomposition $f = x' + f_x$.
- If $f_x = 1$, f has a decomposition $f = x + f_{x'}$.

For example, consider a function $f = a'b'c + a'bd + abd' + ab'c'$. The two cofactors of variable a are $f_{a'} = b'c + bd$ and $f_a = bd' + b'c'$, and $f_{a'} = (f_a)'$ holds. Thus, f has a simple disjunctive decomposition form $f = a \oplus (b'c + bd)$.

To examine whether such decompositions exist, all we have to do is checking the above five conditions for each variable x of f. This task can be done by traversing the OBDD representation of f. For all nodes v labeled with x and the function f^v that v represents, we examine whether f^v satisfies $f^v_{x'} = (f^v_x)'$, $f^v_{x'} = 0$, $f^v_x = 0$, $f^v_{x'} = 1$ or $f^v_x = 1$. The functions $f^v_{x'}$ and f^v_x are represented by the nodes pointed by the edges of v labeled with 0 and 1. Checking whether $f^v_{x'} = (f^v_x)'$ can be done in constant time if negative edges [11] are introduced. Checking whether $f^v_{x'} = 0$, $f^v_x = 0$, $f^v_{x'} = 1$ and $f^v_x = 1$ can also be done in constant time. We can say that these decompositions are also easily detectable.

We have shown two types of simple disjunctive decompositions that are easily detectable. We can expect that many of the other types of simple disjunctive decompositions are also detectable by applying the above two recursively.

4. Constructing new logic representations

In this section, we propose an algorithm that restructures a logic representation (for example, a sum-of-products form or a multi-level form) after a simple disjunctive decomposition is found. This algorithm is applicable to all types of simple disjunctive decompositions, that is, it is not limited to ones described in Section 3.

Suppose that a function f has a simple disjunctive decomposition form $f(X, Y) = h(g(X), Y) = g(X)' \cdot df_0(Y) + g(X) \cdot df_1(Y)$. Let the logic representation of f be F. The algorithm makes new logic representations G and H of g and h by assigning constant values to some variables in F instead of making them from scratch.

4.1. Constructing a representation G of $g(X)$

At first, we show an algorithm that constructs a representation G by assigning constant values δ that satisfy $df_0(\delta) \neq df_1(\delta)$ to variables in Y of F.

1. Calculate $d_P(Y) = df_0(Y)' \cdot df_1(Y)$ and $d_N(Y) = df_0(Y) \cdot df_1(Y)'$.
2. Select minterms $\delta_P, \delta_N \in \{0,1\}^{|Y|}$ which satisfy $d_P(\delta_P) = 1$ or $d_N(\delta_N) = 1$, respectively. At least either δ_P or δ_N exists because $df_0(Y)$ and $df_1(Y)$ are distinct functions. There may be a case when both δ_P and δ_N exist.
3. If δ_P exists, make a new representation G_P by assigning the minterm δ_P to the representation F.
4. If δ_N exists, make a new representation G_N by assigning the minterm δ_N to the representation F and inverting the output.
5. Eliminate useless sub-representations of G_P and/or G_N whose output represent constant functions. Select a better representation which costs less, and let it be G.

The representation G made by the above procedure represents a function $g(X)$. The proof is as follows. Notice that $f(X, Y) = g(X)' \cdot df_0(Y) + g(X) \cdot df_1(Y)$. Because δ_P satisfies $df_0(\delta_P) = 0$ and $df_1(\delta_P) = 1$, G_P represents $g(X)' \cdot 0 + g(X) \cdot 1 = g(X)$. Because δ_N satisfies $df_0(\delta_N) = 1$ and $df_1(\delta_N) = 0$, G_N represents the complement of $g(X)' \cdot 1 + g(X) \cdot 0 = g(X)'$.

Example 1 Figure 2 shows an example. Suppose that a function f is given by a representation $F = a'bc' + (a+b')c$. There exists a simple disjunctive decomposition for a bound set $\{a, b\}$ because $SDF(f, \{a, b\}) = \{c, c'\}$. Let $df_0 = c$ and $df_1 = c'$. Here we make a representation G of g from the representation F. At first, calculate $d_P = df_0' \cdot df_1 = c'$ and select a minterm $\delta_P = 0 (= c')$ that satisfies $d_P(\delta_P) = 1$. Then, make $G_P = a'b$ by assigning δ_P to F. In the

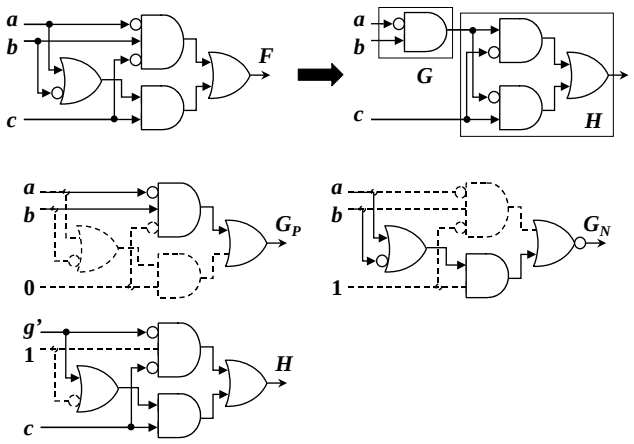

Figure 2. Constructing new representations

same manner, calculate $d_N = df_0 \cdot df_1' = c$ and select a minterm $\delta_N = 1 (= c)$ that satisfies $d_N(\delta_N) = 1$. Then, make $G_N = (a + b')'$ by assigning δ_N to F and inverting the output. The costs of G_P and G_N are the same. Here we select G_P as G.

4.2. Constructing a representation H of $h(g, Y)$

Then we show an algorithm that constructs a representation H by selecting a variable x from the bound set X and assigning a constant value δ that satisfies $g_{x'}(\delta) \neq g_x(\delta)$ to variables in $X - \{x\}$ of F.

1. Select a variable x in X.
2. Calculate $d(X-\{x\}) = g_{x'}(X-\{x\}) \oplus g_x(X-\{x\})$.
3. Select a minterm $\delta \in \{0,1\}^{|X|-1}$ which satisfies $d(\delta) = 1$. For any variable x in X, there exists a minterm δ that satisfies $g_{x'}(\delta) \oplus g_x(\delta) = 1$, unless x is a redundant variable.
4. Make a representation H_I by assigning the minterm δ to the representation F.
5. Because $d(\delta) = 1$, either $g(\delta) = x$ or $g(\delta) = x'$ holds. If $g(\delta) = x$, make a representation H by replacing the variable x in H_I with g. If $g(\delta) = x'$, make a representation H by replacing the variable x in H_I with g'.

The representation H made by the above procedure represents a function $h(g, Y)$. The proof is as follows. Notice that $f = h(g(X), Y)$. When $g(\delta) = x$, H_I represents $h(x, Y)$, and H represents $h(g, Y)$. When $g(\delta) = x'$, H_I represents $h(x', Y)$, and H represents $h((g')', Y) = h(g, Y)$.

```
for each node v {
    if ( v has a multiple fanout or
         v is connected to a primary output ) {
        C = maximum_fanout_free_cone(v);
        newC = simple_disjunctive_decomposition(C);
        if ( cost(newC) < cost(C) ) {
            replace(C, newC);
        }
    }
}
```

Figure 3. Restructuring procedure

Example 2 (Continued from Example 1) As shown in Figure 2, the subfunction is $g = a'b$. Then we make a representation H of h from the representation F. At first, select a variable a from the bound set $\{a, b\}$. Calculate $d = g_{a'} \oplus g_a = b \oplus 0 = b$ and select a minterm $\delta = 1 (= b)$ that satisfies $d(\delta) = 1$. Assign δ to F and make $H_I = a'c' + ac$. Because $g(\delta) = a'$, replace a in H_I with g' and obtain $H = gc' + g'c$.

We can select an arbitrary variable x from X. When the given representation F is of a sum-of-products form, selecting the variable that appears the least is a good heuristic to obtain a smaller sum-of-products form. When the given representation F is a multi-level circuit, selecting the variable that appears the least or is nearest to the output is a good heuristic to obtain a circuit of smaller area or fewer levels.

5. Experiments

We performed experiments to restructure multi-level logic circuits using simple disjunctive decompositions. The procedure is shown in Figure 3, where *maximum_fanout_free_cone*(v) returns the maximum fanout free cone [12] of a node v, *simple_disjunctive_decomposition*(C) makes a new logic representation for a cone C using the techniques described so far, and *replace*(C, $newC$) replaces a cone C with $newC$. Only our procedure does not arrow sharing common logic blocks among maximum fanout free cones, nor simplification of nodes using don't cares. To perform these optimizations, we used SIS [13] after our procedure.

Table 5 shows the experimental results for several multi-level logic circuits [14] shown in the column "Circuit". The column "SIS only" shows the results of the "script.rugged" script of SIS. The column "SDD + SIS" shows the results when our procedure was applied followed by "script.rugged". The subcolumns "literal" show the total numbers of literals in factored forms. The subcolumn "ratio" shows the ratio of the number of literals of "SDD + SIS" to that of "SIS only". The subcolumns "time" give the

Table 1. Experimental results

Circuit	SDD + SIS			SIS only	
name	literal	ratio	time	literal	time
alu2	356	0.99	3.0+ 52.1	361	49.0
apex6	736	0.99	1.7+ 10.0	743	7.5
apex7	248	1.01	0.6+ 2.2	245	2.1
b9	126	1.03	0.7+ 1.3	122	1.1
c8	127	0.91	0.6+ 0.8	139	1.0
cordic	62	0.97	0.9+ 3.4	64	0.5
dalu	938	0.96	4.9+ 69.1	979	128.5
des	3454	0.99	4.2+226.7	3472	218.7
f51m	71	0.78	0.4+ 0.4	91	1.2
frg1	136	1.00	0.9+ 22.1	136	5.2
frg2	731	0.83	3.8+ 20.0	886	35.0
i8	987	0.97	5.6+ 26.7	1015	36.4
k2	1120	0.99	3.1+ 93.5	1135	95.4
lal	104	0.99	0.7+ 1.1	105	1.3
pm1	48	0.96	0.4+ 0.3	50	0.4
rot	672	1.00	1.6+ 12.8	672	12.6
sct	80	1.01	0.6+ 0.9	79	1.3
t481	36	0.04	14.9+ 0.2	881	137.4
term1	137	0.81	2.4+ 4.7	170	8.2
ttt2	189	0.86	0.8+ 1.8	219	2.7
x1	297	1.00	2.5+ 3.8	298	3.3
x3	759	0.97	2.7+ 9.7	785	10.0
x4	389	1.01	1.4+ 4.2	385	4.4
z4ml	36	0.88	0.4+ 0.2	41	0.5
Total	11839	0.91	58.9+568.0	13073	763.7

CPU time in seconds on a Sun Ultra 2 Model 2200. In the column "SDD + SIS", the left hand number of "+" shows that of our procedure and the right hand number shows that of "script.rugged".

We achieved an overall improvement of 9% in the number of literals by using simple disjunctive decompositions. The computation costs for simple disjunctive decompositions were generally small. In the circuits "f51m" and "z4ml", we found many simple disjunctive decomposition forms whose output was a 2-input XOR gate. It seems to be difficult for SIS to find such decomposition forms. There was a drastic improvement for the circuit "t481", where the maximum fanout free cone of the primary output covered the whole circuit and was represented in a tree structure of 2-input XOR/AND gates.

6. Conclusion

We have shown an efficient method to detect simple disjunctive decompositions by detecting symmetric variables or checking variable cofactors. We also have proposed an algorithm to construct a new logic representation from the original representation, which enables us to apply a simple disjunctive decomposition with keeping good structures of the original representation. In experiments, we restructured multi-level logic circuits using these techniques. Although we achieved some improvements, they were limited within each fanout free cone. Thus the result heavily depends on the initial circuit structure. In future work, we want to develop more powerful techniques that restructures logic representations across each fanout free cone.

References

[1] R. L. Ashenhurst, "The Decomposition of Switching Functions," in *Proc. an Int'l Symposium on the Theory of Switching*, pp. 74–116, Apr. 1957.

[2] H. A. Curtis, *A New Approach to the Design of Switching Circuits.* Van Nostrand, 1962.

[3] J. P. Roth and R. M. Karp, "Minimization Over Boolean Graphs," *IBM Journal*, pp. 227–238, Apr. 1962.

[4] S. Chang and M. Marek-Sadowska, "Technology Mapping via Transformations of Function Graphs," in *Proc. Int'l Conf. Computer Design*, pp. 159–162, Oct. 1992.

[5] Y.-T. Lai, M. Pedram, and S. Vrudhula, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," in *Proc. Design Automation Conf.*, pp. 642–647, June 1993.

[6] T. Sasao, "FPGA Design by Generalized Functional Decomposition," in *Logic Synthesis and Optimization* (T. Sasao, ed.), pp. 233–258, Kluwer Academic Publishers, 1993.

[7] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 667–691, Aug. 1986.

[8] H. Sawada, S. Yamashita, and A. Nagoya, "Restricted Simple Disjunctive Decompositions Based on Grouping Symmetric Variables," in *Proc. the Seventh Great Lakes Symposium on VLSI*, pp. 39–44, Mar. 1997.

[9] D. Möller, J. Mohnke, and M. Weber, "Detection of Symmetry of Boolean Functions Represented by ROBDDs," in *Proc. Int'l Conf. Computer-Aided Design*, pp. 680–684, Nov. 1993.

[10] C. R. Edwards and S. L. Hurst, "A Digital Synthesis Procedure Under Function Symmetries and Mapping Methods," *IEEE Trans. Computers*, vol. c-27, pp. 985–997, Nov. 1978.

[11] S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," in *Proc. Design Automation Conf.*, pp. 52–57, jun 1990.

[12] J. Cong and Y. Ding, "On Area/Depth Trade-Off in LUT-Based FPGA Technology Mapping," *IEEE Trans. on VLSI systems*, vol. 2, pp. 137–148, June 1994.

[13] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Memorandum UCB/ERL M92/41, Univ. of California, Berkeley, may 1992.

[14] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide Version 3.0.* MCNC, Jan. 1991.