Symmetry Detection for Incompletely Specified Functions*

Kuo-Hua Wang and Jia-Hung Chen
Dept. of Computer Science and Information Engineering
Fu Jen Catholic University
Taipei, Taiwan, R.O.C.
khwang@csie.fju.edu.tw

ABSTRACT

In this paper, we formulate symmetry detection for incompletely specified functions as an equation without using cofactor computation and equivalence checking. Based on this equation, a symmetry detection algorithm is proposed. This algorithm can simultaneously find non-equivalence and equivalence symmetries. Experimental results on a set of benchmarks show that our algorithm is indeed very effective in solving symmetry detection problem for incompletely specified functions.

Categories and Subject Descriptors: B.6.3 [Design Aids]: Automatic Synthesis, Switching Theory

General Terms: Design, Theory

Keywords: Equivalence Symmetry, Non-Equivalence Symmetry

1. INTRODUCTION

Symmetry detection is to check input symmetries of Boolean functions. It is an important technique in many applications of logic synthesis, physical design, and testing [3]-[10]. In 1960's, many approaches using decomposition chart [1] and truth tables [2] had been proposed. These methods are based on equivalence checking of two-input cofactors, namely the *naive* method. These approaches are very time-consuming and not feasible for large functions. To alleviate this problem, BDD's [11] had been used in symmetry detection[12][13]. [12] proposed some criterions to avoid redundant cofactor computation and thus speed up the computing process. In [13], a very efficient algorithm without computing cofactors was proposed. These approaches are mainly based on BDD's traversal.

Yet, another group of researches without computing cofactors take spectral approach to solve the symmetry detection problem. Many researches had been proposed before [14]-[16]. The major problem of using spectral method is the time complexity is too large while computing spectral coefficients for large functions. In addition, it is not feasible for incompletely specified functions.

Most of the researches mentioned above were proposed to deal with completely specified functions. In this paper, we will propose a symmetry detection method which is not only efficient but also handle incompletely specified functions. Our approach is based on removing non-symmetric sets rather than on directly checking the equivalence of cofactors. In other words, no cofactor computation and equivalence checking will be used in our method.

The remaining of the paper is organized as follows. In Section 2, we will briefly review different types of symmetries for Boolean functions. In Section 3, we formulate the problem of symmetry detection for incompletely specified functions as an equation based on checking pairs of cubes from different care sets. By this equation, a symmetry detection algorithm is proposed. Section 4 will propose a heuristic algorithm to compute maximum symmetries of incompletely specified functions. Some experimental results on a set of benchmarks will be shown in Section 5. Finally, we give a brief conclusion.

2. PRELIMINARIES

Given a Boolean function $f(X)$. f is *totally symmetric* if and only if it is invariant under any permutation of input set X. f is *partially symmetric* if and only if it is invariant under the permutation of a subset $T \subset X$. We call T a *symmetric set* of f. T is a *symmetric pair* of f if its size is 2. For completely specified functions, symmetric pair is an equivalence relation. That is, all symmetric sets can be generated directly from symmetric pairs. However, this statement is not true for incompletely specified functions.

The *cofactor* of $f(X)$ with respect to (w.r.t.) a variable $x_i \in X$ is $f_{x_i} = f(x_1, \cdots, x_i = 1, \cdots, x_n)$. The cofactor of $f(x)$ w.r.t. a literal $\bar{x_i}$ is $f_{\bar{x_i}} = f(x_1, \cdots, x_i = 0, \cdots, x_n)$. Given any two inputs x_i and x_j in X. There are four cofactors of f w.r.t. x_i and x_j, i.e., $f_{\bar{x_i}\bar{x_j}}$, $f_{\bar{x_i}x_j}$, $f_{x_i\bar{x_j}}$, and $f_{x_ix_j}$. Different types of symmetries can be defined according to the equality of two cofactors among them. For the case of $f_{\bar{x_i}x_j} = f_{x_i\bar{x_j}}$, it is *non-equivalence symmetry* and denoted as $NE(x_i, x_j)$. While *equivalence symmetry* (E) and *single variable symmetry* (SV) are defined with regard to $f_{\bar{x_i}\bar{x_j}} = f_{x_ix_j}$ and $f_{\bar{x_i}x_j} = f_{x_ix_j}$, respectively.

Consider NE and E symmetries, it is clear that $NE(x_i, \bar{x_j})$ and $E(x_i, x_j)$ are identical. In this paper, we will use (x_i, x_j) and $(x_i, \bar{x_j})$ to represent $NE(x_i, x_j)$ and $E(x_i, x_j)$, respectively. Besides, (x_j, x_i) will be normalized to (x_i, x_j) if $i < j$.

*This work was supported in part by the R.O.C. National Science Council under Grant NSC92-2220-E-030-002.

DAC 2004, June 7–11, 2004, San Diego, California, USA.

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

Figure 1: The Karnaugh map of $f = (f^{on}, f^{off})$.

3. THE SYMMETRY DETECTION ALGORITHM

In this section, we first show how to detect symmetries of incompletely specified functions based on checking pairs of cubes from different care sets. Then a symmetry detection algorithm and some implementation issues will be addressed.

3.1 Symmetry Detection with Don't Cares

An incompletely specified function f is a Boolean function with don't cares. It involves three sets: the on-set (f^{on}), the off-set (f^{off}), and the don't-care set (f^{dc}). In this paper, we will denote an incompletely specified function as $f = (f^{on}, f^{off})$. Consider two functions f and g w.r.t. input set X. They are *consistent*, denoted as $f \cong g$, if and only if $f^{on} \cap g^{off} = \emptyset$ and $f^{off} \cap g^{on} = \emptyset$. They are *equivalent*, denoted as $f \equiv g$, if and only if $f^{on} = g^{on}$ and $f^{off} = g^{off}$.

By the definitions of *consistency* and *equivalence*, *weak* and *strong NE-symmetries* can be defined below. The weak and strong equivalence (E) and single variable (SV) symmetries can be defined in a similar way.

DEFINITION 3.1. *A function $f(X)$ is weakly NE symmetric in two variables $x_i, x_j \in X$ if and only if $f_{\bar{x_i}x_j} \cong f_{x_i\bar{x_j}}$. It is denoted as $WNE(x_i, x_j)$. $f(X)$ is strongly NE symmetric in two variables $x_i, x_j \in X$ if and only if $f_{\bar{x_i}x_j} \equiv f_{x_i\bar{x_j}}$. It is denoted as $SNE(x_i, x_j)$.* □

For ease of explaining our method, we will use *minterm pair* to show our idea on symmetry detection. Given a minterm m_i w.r.t. the input set $X = \{x_1, \cdots, x_n\}$. The *weight* of m_i, denoted as $Weight(m_i)$, is the number of 1 bits in the encoded binary number i. The *Hamming distance* between minterms m_i and m_j is the number of disjoint input values between them. The distance and disjoint input set are denoted as $Distance(m_i, m_j)$ and $Disjoint(m_i, m_j)$, respectively. The same definitions can be applied to cube pairs.

Now, we will explain our method using minterm pair. Consider a function f shown in Fig. 1. We find that $WNE(x_1, x_2)$ can't hold since $f_{\bar{x_1}x_2}$ and $f_{x_1\bar{x_2}}$ are inconsistent in the minterm pair $P = (m_4, m_8)$. To examine P, $Weight(m_4) = Weight(m_8)$ and $Distance(m_4, m_8) = 2$. $WE(x_1, x_2)$ can't hold alike. With the above observation, Theorem 3.1 can be derived directly.

THEOREM 3.1. *Given a function $f(X) = (f^{on}, f^{off})$ and a minterm pair (m_a, m_b), where $m_a \in f^{on}$ and $m_b \in f^{off}$. $\{x_i, x_j\}$ can't be a NE (E) symmetric set of f if and only if $Weight(m_a) = Weight(m_b)$ ($Weight(m_a) \neq Weight(m_b)$) and $Disjoint(m_a, m_b) = \{x_i, x_j\}$.* □

Now consider a cube pair $u \in f^{on}$ and $v \in f^{off}$. In order to remove non-symmetric sets, three rules based on Theorem 3.1 are applied to check this pair of cubes.

Rule 1: If $Distance(u, v) > 2$, no non-symmetric set will be removed.

Rule 2: If $Distance(u, v) = 2$, one non-symmetric set will be removed.

Rule 3: If $Distance(u, v) = 1$, multiple non-symmetric sets will be removed.

Rule 1 is clear that cube pair (u, v) can't involve any minterm pairs with distance 2. Rule 2 means that it can only involve one type of minterm pairs with the same disjoint input variables. Let $x_i(u)$ denotes the value of input x_i in cube u. The following example illustrates Rule 3.

EXAMPLE 3.1. Consider two cubes $u = \bar{x_1}\bar{x_2} \in f^{on}$ and $v = x_1\bar{x_3} \in f^{off}$ w.r.t. input set $X = \{x_1, x_2, x_3, x_4\}$. The $Distance(u, v) = 1$ and $Disjoint(u, v) = \{x_1\}$. So Rule 3 is used to remove non-symmetric sets. The remaining possible disjoint inputs are x_2, x_3, and x_4. W.r.t. inputs x_2 and x_3, $E(x_1, x_2)$ and $NE(x_1, x_3)$ are removed since $x_1(u) = x_2(u)$ and $x_1(v) \neq x_3(v)$, respectively. As to the input x_4, $NE(x_1, x_4)$ and $E(x_1, x_4)$ are removed since $x_4(u)$ and $x_4(v)$ are don't cares, i.e., both disjoint situations exist for x_4. □

From the above description, all symmetric sets of $f = (f^{on}, f^{off})$ can be detected as follows: "*Apply Rule 1, 2 and 3 to each cube pair (u, v), where $u \in f^{on}$ and $v \in f^{off}$, respectively. After all cube pairs have been checked, then the sets not removed are weakly symmetric sets of f.*"

3.2 Removing Non-Symmetric Sets by Cube Pairs

We give some notations as follows. Let u and v be binary-valued cubes w.r.t. input set X. *1-lit(u)* (*0-lit(u)*) is a subset of X, where only literals of u in uncomplemented (complemented) form are in the subset. $x_i(u)$ denotes the value of x_i in cube u. *Non-Symmetries*(u, v) denotes the non-symmetric sets removed by cube pair (u, v), where $u \in f^{on}$ and $v \in f^{off}$, respectively.

By Rule 1, 2, and 3, Lemma 3.1 is proposed. Theorem 3.2 formulates symmetry detection for incompletely specified functions as an equation using set operations.

LEMMA 3.1. *If $Distance(u, v) > 2$, then* *Non-Symmetries*$(u, v) = \emptyset$.
If $Distance(u, v) = 2$ and $Disjoint(u, v) = \{x_i, x_j\}$, then

$$\textit{Non-Symmetries}(u,v) = \begin{cases} \{(x_i, x_j)\} & : \text{ if } x_i(u) \neq x_j(u) \\ \{(x_i, \bar{x_j})\} & : \text{ if } x_i(u) = x_j(u). \end{cases} \quad (1)$$

If $Distance(u, v) = 1$, then

$$\textit{Non-Symmetries}(u,v) = \begin{cases} A \times (B \cup \overline{C}) & : \text{ if } x_i(u) = 0 \\ A \times (\overline{B} \cup C) & : \text{ if } x_i(u) = 1 \end{cases} \quad (2)$$

, where $A = Disjoint(u, v) = \{x_i\}$, $B = ((X - 0\text{-}lit(u)) \cap (X - 1\text{-}lit(v))) - A$, and $C = ((X - 1\text{-}lit(u)) \cap (X - 0\text{-}lit(v))) - A$. $\bar{B}$ ($\bar{C}$) includes all inputs of B (C) in complemented form. □

THEOREM 3.2. *Given $f = (f^{on}, f^{off})$. The* equivalence *and non-equivalence symmetric sets ψ of f can be formulated as*

$$\psi = \textbf{Sym-totality} - \bigcup_{\substack{i=1,\ldots,|f^{on}| \\ j=1,\cdots,|f^{off}|}} \textit{Non-Symmetries}(u_i, v_j) \quad (3)$$

```
Algorithm Compute-Symmetries(f(X))
Input: f(X) = (f^on, f^off);
Output: return ψ or ∅;
Begin
    ψ = Sym-totality;
    S = ∅;
    for each cube u_i ∈ f^on do
        if (u_i is redundant) then continue;
            ω = Non-Symmetries(u, v);
            ψ = ψ − ω;
            if (ψ is ∅) then return ∅;
        endfor
        S = Non-Symmetric-Inputs(ψ);
        if (S is changed) then Reduction(f^off, S);
    endfor
    return ψ;
End
```

Figure 2: The *Compute-Symmetries* **algorithm.**

, where **Sym-totaliy** $= \{(x_i, x_j), (x_i, \bar{x_j}) | i = 1, \cdots, |X| - 1, j = i+1, \cdots, |X|\}$*,* $|f^{on}|$ *and* $|f^{off}|$ *are the numbers of cubes for* f^{on} *and* f^{off}*,* $u_i \in f^{on}$ *and* $v_j \in f^{off}$*, respectively.* □

3.3 Implementation Issues

By Equation (3), we must check all cube pairs to remove non-symmetric sets. However, many non-symmetric sets removals are redundant and thus increase the computing time. For example, if input x_i had been found that it can't be symmetric with the other inputs, then any removal involving x_i is redundant. By such an observation, let S be such inputs computed by procedure *Non-Symmetric-Inputs*. A cube in on-set and off-set is *redundant* if it only involves inputs from S. Redundant cubes must be deleted during the computation process. The procedure *Reduction* is called to remove redundant cubes. Our experimental results show that many redundant cubes are found during the computing process. The above symmetry detection algorithm is implemented as procedure *Computing-Symmetries* shown in Fig. 2.

4. MAXIMUM SYMMETRIES OF INCOMPLETELY SPECIFIED FUNCTIONS

The difficulties of finding large symmetric sets for incompletely specified functions is due to that weak symmetry is not an equivalence relation. Large symmetric sets can't be deduced from weakly symmetric pairs directly. In this section, we propose a heuristic algorithm *Compute-Maximum-Symmetry* to solve this problem. It is shown in Fig. 3.

The procedure *Compute-Maximum-Symmetry* will return a derived function $\tilde{f}$ and a minimum-sized partition P of X, where $\tilde{f}$ is strongly symmetric in P. This algorithm uses strong symmetric sets as the initial partition of X. Then based on weak symmetry checking, these symmetric sets can grow gradually by adding inputs to them. In this algorithm, the procedure *Make-Strong-Symmetries* transforms a weakly symmetric set (x_i, x_j) of f into a strongly symmetric one. It returns a function $\tilde{f}$ derived from f by assigning don't cares to f.

Initially, weakly symmetric sets (WSS) are computed first. Strongly symmetric sets (SSS) can be computed from WSS by calling the procedure *Modified-Compute-Symmetries*

```
Algorithm Compute-Maximum-Symmetries(f, X)
Input: f(X) = (f^on, f^off);
Output: return (f̃, P);
Begin
    WSS = Compute-Symmetries(f);
    SSS = Modified-Compute-Symmetries(f, WSS);
    P = Equivalence-Classes(SSS);
    (x_i, Y) = Maximum-Set(f, WSS, P);
    f̃ = f;
    while (there exists such an input x_i) do
        g = f̃;
        for each x_j ∈ Y do
            f̃ = Make-Strong-Symmetry(f̃, x_i, x_j);
        endfor
        WSS = Revise-WSS(WSS, g, f̃);
        SSS = Modified-Compute-Symmetries(f̃, WSS);
        P = Equivalence-Classes(SSS);
        (x_i, Y) = Maximum-Set(f̃, WSS, P);
    endwhile
    return (f̃, P);
End
```

Figure 3: The *Compute-Maximum-Symmetries* **algorithm.**

which only deals with cube pairs from f^{dc} and $f^{on} \cup f^{off}$. *Equivalent-Classes* procedure is used to find the initial partition P of X. The reason we use this initial partition is that no don't cares will be assigned by P. Following this, the procedure *Maximum-Set* is called to search an input x_i and a symmetry set Y in P such that $Y \cup \{x_i\}$ can form a larger weakly symmetric set. The heuristic behind *Maximum-Set* procedure is to select the largest Y among those strong symmetric sets in the partition P and x_i such that minimum don't cares will be assigned. *Make-Strong-Symmetry* procedure is called to compute the derived function $\tilde{f}$. Finally, *Revise-WSS* is called to revise *WSS*. Repeat the above process until no x_i and Y can be grouped to form a larger symmetric set w.r.t. $\tilde{f}$.

5. EXPERIMENTAL RESULTS

We have implemented the proposed symmetry detection algorithm in C language on SUN-Blade 1000 station. The implementation platform is based upon *Espresso* package in SIS [17]. Our algorithm can be applied both to completely specified and incompletely specified functions. To demonstrate the efficiency of our algorithm, MCNC benchmarking circuits have been tested in our experiments.

The first experiment was performed on completely specified functions. We also implemented the naive method using cube notations and compared it with our method. Table 1 shows the experimental results. The columns with labels **#in** and **#out** show the numbers of inputs and outputs of circuits, respectively. The column **#avg** indicates the average number of non-symmetric sets removed by each cube pair. The column **CPU** shows the running time in seconds. The columns **naive** and **ours** show the running time of naive method and ours. The column **#symm** gives the information on symmetries of benchmarking circuits: $n(m)$ means that there are n symmetric sets of m input variables. The symbol **-** in this column indicates the circuit doesn't have any symmetric sets. Summing up the total running time of all benchmark circuits, the running time is 2.12 second. It shows that our method is very efficient for large functions.

Table 1: Benchmarking Results for Completely Specified Functions

circuit	#in	#out	#avg	CPU		#symm
				naive	ours	
9sym	9	1	2	0.04	0.00	1(9)
alu4	14	8	8	0.00	0.08	-
apex1	45	45	35	0.65	0.01	-
apex2	39	3	40	12.87	0.31	1(3), 3(2)
apex3	54	50	36	1.43	0.01	-
apex5	117	88	204	682.20	0.19	2(5)
cordic	23	2	24	1.06	0.58	2(4), 3(3)
duke2	22	29	20	0.03	0.01	-
ex4	128	28	148	562.20	0.09	1(87), 14(2)
e64	65	65	87	0.00	0.50	-
misex2	25	18	39	0.03	0.00	1(6), 1(2)
pdc	16	40	6	0.30	0.15	-
seq	41	35	35	4.67	0.02	2(2)
spla	16	46	6	0.18	0.02	-
t481	16	1	15	0.09	0.14	8(2)
vg2	25	8	29	0.08	0.01	2(2), 21(1)
Total				1265.83	2.12	
Ratio (%)				100.00	0.2	

The second experiment is conducted for incompletely specified functions. To show the different size of don't care set, we perform this experiment by removing the cubes from on-set and off-set. Table 2 shows the experimental results. The column **-80%** and **-40%** show the results of cases, where 80% and 40% cubes in care sets are removed, respectively. The column **#max** gives the maximum number of equivalence and non-equivalence symmetric pairs, i.e., $n \times (n-1)$, where n is the number of inputs. The columns **W** and **S** show the number of weakly and strongly symmetric pairs, respectively. In average, the ratios of **W** to **#max** are 77% and 29% for the cases of removing 80% and 40% cubes, respectively. For strong symmetry, the ratios are 8% and 5% w.r.t. the cases of removing 80% and 40% cubes, respectively. It shows that few strongly symmetric sets exist for incompletely specified functions.

The third experiment is to find the possible maximum symmetries for incompletely specified functions. 60% don't cares are removed from the care-sets of benchmarking circuits. Table 3 shows the experimental results. The columns **initial** and **final** give the initial strongly symmetric sets and the maximum symmetric sets computed by our algorithm, respectively. It shows our algorithm can effectively find large symmetric sets.

6. CONCLUSION

In this paper, we propose an algorithm to detect symmetries for incompletely specified functions. Our algorithm can simultaneously find non-equivalence and equivalence symmetries. Experimental results show that our algorithm is indeed very efficient for many benchmarking circuits. Future works will use BDD's instead of cube notations to represent Boolean functions in our algorithm.

Table 2: Weak Symmetry vs. Strong Symmetry

circuit	#in	#max	80%		40%		CPU
			W	S	W	S	
alu4	14	182	182	0	0	0	4.41
apex1	45	1980	423	5	31	1	8.55
apex3	54	2862	1894	56	158	0	2.52
apex5	117	13572	13572	433	4417	16	18.14
cordic	23	506	68	8	53	6	23.11
ex4	128	16256	11370	2564	6018	1902	2.97
misex2	25	600	397	40	97	8	0.03
spla	16	240	127	0	43	0	17.62
t481	16	240	58	0	19	0	2.02
vg2	25	600	600	2	16	2	0.32
Total		37038	28691	3108	10852	1935	79.69
Ratio(%)		100	77	8	29	5	

Table 3: Benchmarking Results for Maximum Symmetries

circuit	#in	initial	final	CPU
apex1	45	1(3), 1(2)	1(5), 3(2)	4.45
apex3	54	54(1)	1(3), 5(2)	2.33
cordic	23	2(3)	1(5), 2(4), 3(3)	695.78
duke2	22	22(1)	1(3), 2(2)	0.28
x4	128	1(51), 10(2)	1(86), 1(3),10(2)	329.65
misex2	25	1(6), 1(4)	1(12), 1(4), 1(2)	0.18
pdc	16	16(1)	3(2)	3.71
seq	41	2(2)	1(4), 1(3), 1(2)	7.95
spla	16	16(1)	1(4), 1(3),1(2)	5.43
t481	16	1(2)	1(4), 6(2)	2.33
vg2	25	2(2)	1(7), 1(4), 2(2)	2.95
Total				1055.04

7. REFERENCES

[1] A. Mukhopadhyay, "Detection of total or partial symmetry of a switching function with the use of decomposition charts," *IEEE Trans. Electron. Comput.*, vol. EC-12, pp. 535-557, Oct. 1963.

[2] D.L. Dietmeyer and P.R. Schneider, "Identification of symmetry, redundancy, and equivalence of Boolean functions," *IEEE Trans. Electron. Comput.* vol. 16, pp. 804-817, Dec. 1967.

[3] B.G. Kim and D.L. Dietmeyer, "Multilevel logic synthesis of symmetric switching functions," *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 436-446, April 1991.

[4] R. Drechsler and B. Becker, "Sympathy: fast exact minimization of fixed polarity Reed-Muller expressions for symmetric functions," *IEEE Trans. on Computer-Aided Design*, Vol. 16, pp. 1-5, Jan. 1997.

[5] V.N. Kravet and K.A. Sakallah, "Constructive library-aware synthesis using symmetries," in *Proc. European Design Test Conf.*, 2000, pp. 208-213.

[6] C. Tsai and M. Marek-Sadowska, "Boolean matching using generalized Reed-Muller forms," in *Proc. Design Automation Conf.*, pp. 339-344, June, 1994.

[7] F.A. Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah, "Solving difficult SAT instances in the presence of symmetry," in *Proc. Design Automation Conf.*, pp. 731-736, 2002.

[8] C.W. Chang, Bo Hu, and M. Marek-Sadowska, "In-place delay constrained power optimization using functional symmetries," in *Proc. European Design Test Conf.*, pp. 377-382, 2001.

[9] S. Chakrabarti, S. Das, D.K. Das, and B.B. Bhattacharya, "Synthesis of symmetric functions for path-delay fault testability," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 1076-1081, Sept. 2000.

[10] C. Scholl, S. Melchior, G. Hotza, and P. Molitor, "Minimizing ROBDD sizes of incompletely specified functions by exploiting strong symmetries," in *Proc. European Design Test Conf.*, Mar. 1997, pp. 229-234.

[11] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 677-691, 1986.

[12] C. Scholl, D. Moller, P. Molitor, and R. Drechsler, "BDD minimization using symmetries," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 81-100, Feb. 1999.

[13] Alan Mishchenko, "Fast computation of symmetries in Boolean functions," *IEEE Trans. Computer-Aided-Design*, vol. 18, pp. 1588-1593, Nov. 2003.

[14] S. Rahardja, B.J. Falkowski, "Symmetry conditions of Boolean functions in complex Hadamard transform," *Electronics Letters*, vol. 34, pp. 1634-1635, Aug. 1998.

[15] C.C. Tsai and M. Marek-Sadowska, "Generalized Reed-Muller forms as a tool to detect symmetries," *IEEE Trans. Comput.*, vol. 45, pp. 33-40, Jan. 1996.

[16] B.J. Falkowski and S. Kannurao, "Skew symmetry detection using the Walsh spectral coefficients," in *Proc. International Symposium on Cirucuit and System*, 2000, vol. 2, pp. 321-324.

[17] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis", *Electronics Research Laboratory*, Memorandum No. UCB/ERL M92/41, 4 May 1992.