VLSI Design, 2002 Vol. 14 (3), pp. 259–271

Taylor & Francis
Taylor & Francis Group

Boolean Matching Filters Based on Row and Column Weights of Reed–Muller Polarity Coefficient Matrix

CHIP-HONG CHANG and BOGDAN J. FALKOWSKI*

Nanyang Technological University, School of Electrical and Electronic Engineering, Block S1, Nanyang Avenue, Singapore 639798

(Received 11 May 2000; Revised 15 July 2000)

In this article, we have shown, by means of the EXOR Ternary Decision Diagram that the number of literals and product terms of the Fixed Polarity Reed–Muller (FPRM) expansions can be used to fully classify all Boolean functions in NP equivalent class and NPN equivalent class, respectively. Efficient graph based algorithms to compute the complete weight vectors have been presented. The proof and computation method has led to the derivation of a set of characteristic signatures that has low probability of aliasing when used as the Boolean matching filters in library mapping.

Keywords: Boolean matching; Fixed polarity Reed–Muller expansion; Algebraic decision diagram; EXOR ternary decision diagram; NPN classification

INTRODUCTION

Classification of switching functions is an important problem in logic design used for the development of Universal Logic Modules [9–11] and for cell library binding[4,6]. Cell library binding, also frequently called technology mapping, is the process of transforming arbitrary free logic circuits into bound logic circuits where the interconnection of components are the instances of basic elements from a given library. This step is extremely important for both standard-cell and array-based digital semi-custom circuit design. After performing cell library binding, a new complete structural representation of an original free logic circuit is available and serves as an interface to physical design tools in the next step of the computer-aided design process. Frequently, cell library binding uses some type of classification of switching functions as a tool to match efficiently the library instances with free logic circuits [3,4,6,7,18].

There exist different classification schemes. PN classification, the method which involves *Permutation* and/or *Negation* of input variables was first introduced by Staff of the Harvard Computation Laboratory [14] to categorize switching functions. The method was later revised to *NPN* classification [3,7,10,11,14] which, in addition to Negation and Permutation of input variables, involves the *Negation* of the overall function. Another important algebraic classification is the Chow parameter classification [7,11] which involves extensive investigation on linearly separable functions.

Spectral methods for classification of switching functions by verification of Walsh spectral coefficients [7,11] or Complex Hadamard Walsh spectral coefficients [13] have been developed. The latter method is more efficient as it does not require any rearrangement of the spectral coefficients once the coefficients have been calculated. Another classification scheme is based on the Reed–Muller transforms [3,5,18]. In the first work [5] based on this approach, the Reed–Muller weight vector is used to find PN equivalence relation for Boolean functions. In the latest article [18] NPN equivalent classes for three and four-variable functions are uniquely identified where majority of the classes have a single Fixed Polarity Reed–Muller (FPRM) expansion as their representative.

A formal proof of the invariant properties of FPRM expansion has been provided by the authors using the subnumber operation [8]. In this article, an alternative proof based on the EXOR Ternary Decision Diagram [15] has been shown to provide a greater insight into graph based manipulation of the FPRM weight vectors and their application as Boolean matching filters. The derivation of proof has led to the efficient methods of computation of the weight vectors in term of both the number of product and literals. The method relies on the recursive transformation from an EXOR Ternary Decision Diagram

*Corresponding author. Tel.: +65-7904521. Fax: +65-7912687. E-mail: efalkowski@ntu.edu.sg

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/10655140290011069

representation of Boolean vector into an Algebraic Decision Diagram capable of storing integer valued vector [15]. The proposed method has overcome the constraint of memory overhead as compared with the natural method of direct vector operations.

In library based technology mapping, matching of the intermediate circuit functions against the library of existing cell functions is a computational expensive operation. For each pair of functions being matched, it involves searching of a correspondence of variables and a polarity value for each variable. The NPN invariant properties of the FPRM weight vector is exploited in this paper to derive various low aliasing Boolean matching filters that could efficiently determine if two Boolean functions are certainly not equal or equal with a low probability of uncertainty. Using the entire weight vector as characteristic signature is guarantee to be aliasing free, but such an approach suffers from exponential complexity of both time and space. In this paper, we propose various characteristic signatures based on a selected set of weights at the functional and variable (cofactor) levels in order to obtain a linear time and space complexity with relatively low number of aliases. By transforming the resulting characteristic signature into a character string representation, considerable speed up can be achieved as each matching operation is simply a fast string comparison. The string length can be further significantly compressed by applying run-length coding [17] on repetitive patterns.

BASIC DEFINITIONS

An n-variable Boolean function can be expressed as a *canonical Reed–Muller expansion* [1–3,8,9,15,16,18] of 2^n terms as follows:

$$F(x_n, x_{n-1}, \ldots, x_1) = \bigoplus_{j=0}^{2^n-1} a_j \prod_{i=1, j_i=1}^{n} x_i^{\omega_i} \qquad (1)$$

where $\oplus$ denotes the modulo-2 addition, $a_j \in (0, 1)$ is called a Reed–Muller coefficient and $j_i \in (0, 1)$ is the i-th bit of the binary representation of j, with j_i, being the least significant digit. $\omega_i \in (0, 1)$ is the polarity bit of the variable x_i. $x_i^{\omega_i} = x_i$ when $\omega_i = 0$, and $x_i^{\omega_i} = \bar{x_i}$ when $\omega_i = 1$. When each literal $(x_i^{\omega_i}, i = 1, 2, \ldots, n)$ throughout the expression in Eq. (1) has consistent polarity bit value, such an expression is known as a *Fixed Polarity Reed–Muller* (*FPRM*) expansion [1–3,8,9,15,16,18].

DEFINITION 1 *The polarity number, denoted by ω, is defined as the decimal equivalence of a binary n-tuple formed by writing a 0 or a 1 for each literal, according to whether it appears in affirmation or negation in all the terms of the FPRM expansion.*

DEFINITION 2 *A polarity vector, denoted by A^{ω}, is an ordered set of all 2^n FPRM expansion coefficients $[a_0 a_1 \ldots a_{2^n-1}]$ in some chosen polarity number ω.*

DEFINITION 3 *A Polarity Coefficient matrix PC(F) is a $2^n \times 2^n$ binary matrix where the vector in row i is the polarity vectory A^i.*

DEFINITION 4 *The weight w_p, is the number of product terms of a FPRM expansion [3,5,15]. w_p is also the number of non-zero elements in the polarity vector A^{ω}. The weight w_l, is the total number of literals of a FPRM expansion. w_l is also the sum of the Hamming weights of the indices of the non-zero elements in the polarity vector A^{ω}, where the Hamming weight of an integer is the number of "1's" in its binary representation.*

DEFINITION 5 *An ordered set of all 2^n weights w_p (or w_l) corresponding to the 2^n polarity vectors of a Boolean function F, arranged in ascending order of their polarity numbers ω, is denoted by the symbol W_p (or W_l).*

DEFINITION 6 *An EXOR Ternary Decision Diagram (ETDD) is a rooted, directed acyclic graph with vertex set V and edge set E [15]. Two types of vertices are defined. A non-terminal vertex v has as attributes an index,* $\text{index}(v) \in \{1, 2, \ldots, n\}$, *and three children* $\text{low}(v)$, $\text{high}(v)$ *and* $\text{xor}(v) \in V$. *A terminal vertex v has as attribute a constant value,* $\text{value}(v) \in \boldsymbol{B}$. *Each non terminal has three outgoing edges, e_0, e_1, $e_2 \in E$ with* $\text{value}(e_0) = 0$, $\text{value}(e_1) = 1$ *and* $\text{value}(e_2) = 2$, *directed towards* $\text{low}(v)$, $\text{high}(v)$ *and* $\text{xor}(v)$, *respectively. In addition, $f_{\text{xor}(v)} = f_{\text{low}(v)} \oplus f_{\text{high}(v)}$, where f_v represents the function of the vertex v. For any non terminal vertex $v \in V$, if* $\text{child}(v) \in \{\text{low}(v), \text{high}(v), \text{xor}(v)\}$ *is also a non terminal vertex, then* $\text{index}(\text{child}(v)) > \text{index}(v)$.

The ETDD is a *Reduced Ordered ETDD* if there exists no isomorphic subgraph rooted by distinct vertices, and there is no vertex v with $low(v) = high(v)$. In a reduced ordered ETDD, if v is the predecessor of the terminal **1**-vertex, and $index(v) < n$, the **1**-vertex, can be expanded to a non terminal vertex u, with $index(u) = index(v) + 1$, $low(u) = high(u) =$ **1**-vertex and $xor(u) =$ **0**-vertex. Input and output negation edge attributes can be introduced to further reduce the number of vertices. The input and output negation attributes are denoted by the symbols "•" and "◆", respectively; on the incident edge of the vertex. If a vertex v contains an input negation attribute, then the same vertex will represent $\bullet v$ with $low(\bullet v) = high(v)$, $high(\bullet v) = low(v)$ and $xor(\bullet v) = xor(v)$. If v contains an output negation attribute, then $\blacklozenge v$ will have $low(\blacklozenge v) = \blacklozenge low(v)$, $high(\blacklozenge v) = \blacklozenge high(v)$ and $xor(\blacklozenge v) = xor(v)$.

ALGEBRAIC CLASSIFICATION BY WEIGHT VECTORS OF FPRM EXPANSIONS

The principal algebraic classification method [3,7,10,11,13,14] considers the following operations, taken individually or collectively;

(a) negation (N) of one more input variables of the functions,
(b) permutation (P) of two or more of the input variables of the functions, and
(c) negation (N) of the output of the functions.

Functions that are invariant under the above three operations belong to the same *NPN-equivalent* class. Functions that are invariant under combinations of operations (a) and (b) only are in the same *PN-equivalent* class. The NPN invariant properties of the FPRM weight vectors have been proven by the subnumber operations in Ref. [8]. Here, we will show an efficient method to calculate the FPRM weight vectors and verify their NPN invariant properties based on the ETDD data structure.

The following expansions relate the function represented by vertex v with variable x_i as its top variable, and the functions represented by its children:

$$f_v = \bar{x}_i f_{\text{low}(v)} \oplus x_i f_{\text{high}(v)} \tag{2}$$

$$f_v = \bar{x}_i f_{\text{xor}(v)} \oplus f_{\text{high}(v)} \tag{3}$$

$$f_v = x_i f_{\text{xor}(v)} \oplus f_{\text{low}(v)} \tag{4}$$

Equations (2)–(4) correspond to the Shannon's, negative Davio's and positive Davio's decompositions, respectively [5,15]. From Eq. (3), we observe that

$$\begin{aligned} w_p(f_v) &= w_p(f_{\text{xor}(v)}) + w_p(f_{\text{high}(v)}) \text{ and} \\ w_l(f_v) &= w_p(f_{\text{xor}(v)}) + w_l(f_{\text{xor}(v)}) + w_l(f_{\text{high}(v)}) \end{aligned} \tag{5}$$

Similarly, from Eq. (4)

$$\begin{aligned} w_p(f_v) &= w_p(f_{\text{xor}(v)}) + w_p(f_{\text{low}(v)}) \text{ and} \\ w_l(f_v) &= w_p(f_{\text{xor}(v)}) + w_l(f_{\text{xor}(v)}) + w_l(f_{\text{low}(v)}) \end{aligned} \tag{6}$$

```
weight(v, ω)
{
    if (index(v) > n) return (value(v), 0);
    (w_p0, w_l0) = weight(low(v), ω);
    (w_p1, w_l1) = weight(high(v), ω);
    (w_p2, w_l2) = weight(xor(v), ω);
    i = index of top variable of v;
    if (ω_i == 0) return (w_p0 + w_p2,  w_p2 + w_l0 + w_l2);
    else (w_p1 + w_p2,  w_p2 + w_l1 + w_l2);
}
```

FIGURE 1 Calculation of w_p and w_l from ETDD.

```
weight_vector(v)
{
    if (index(v) > n) return (value(v), 0);
    (W_p0, W_l0) = weight_vector(low(v));
    (W_p1, W_l1) = weight_vector(high(v));
    (W_p2, W_l2) = weight_vector(xor(v));
    return ( [ W_p0 + W_p2 ], [ W_p2 + W_l0 + W_l2 ] );
             [ W_p1 + W_p2 ]  [ W_p2 + W_l1 + W_l2 ]
}
```

FIGURE 2 Calculation of W_p and W_l from ETDD.

Since the FPRM expansion of a given polarity number can be obtained by positive and negative Davio's decompositions, the weights w_p and w_l of an FPRM expansion can be obtained recursively by applying Eq. (5) if $\omega_i = 1$ and Eq. (6) if $\omega_i = 0$ for every vertex v with top variable x_i. The algorithm **weight** for the calculation of the number of products (w_p) and literals (w_l) for any polarity number ω from the ETDD is shown in Fig. 1. The algorithm can be modified to compute the entire weight vectors W_p and W_l. This is shown in Fig. 2. In Procedure **weight_vector**, W_{pi} and W_{li} $(i = 0, 1, 2)$ are column vectors of dimension $2^{n\text{-index}(v)}$ at vertex v. When a constant is returned from the calling routine at vertex v, it will be expanded into a column vector of dimension $2^{n\text{-index}(v)}$ with all entries filled with that value. It should be noted that a **1**-vertex contributes one product count but zero literal count. If input negation attribute is used, some computational saving can be achieved. The weight vector $W_p(\bullet v)$ can be calculated by swapping the upper and lower halves of the vector $W_p(v)$. The same operation can be applied to $W_l(v)$ to obtain $W_l(\bullet v)$.

Example 1 The ETDD of the function $f = \sum m(2, 3, 4, 5, 6)$ is shown in Fig. 3. The weight vector W_p and W_l are computed to the procedure **weight_vector** as follows:

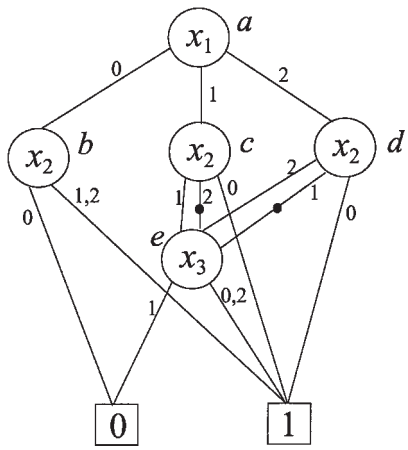

FIGURE 3 ETDD of the function $\Sigma m(2, 3, 4, 5, 6)$.

$$W_p(b) = \begin{bmatrix} W_p(0) + W_p(1) \\ W_p(1) + W_p(1) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix},$$

$$W_l(b) = \begin{bmatrix} W_p(1) + W_l(0) + W_l(1) \\ W_p(1) + W_l(1) + W_l(1) \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$W_p(e) = \begin{bmatrix} W_p(1) + W_p(1) \\ W_p(0) + W_p(1) \end{bmatrix} = \begin{bmatrix} 1+1 \\ 0+1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix},$$

$$W_l(e) = \begin{bmatrix} W_p(1) + W_l(1) + W_l(1) \\ W_p(1) + W_l(0) + W_l(1) \end{bmatrix} = \begin{bmatrix} 1+0+0 \\ 1+0+0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$W_p(c) = \begin{bmatrix} W_p(1) + W_p(\bullet e) \\ W_p(e) + W_p(\bullet e) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 3 \\ 3 \\ 3 \end{bmatrix},$$

$$W_l(c) = \begin{bmatrix} W_p(\bullet e) + W_l(1) + W_l(\bullet e) \\ W_p(\bullet e) + W_l(e) + W_l(\bullet e) \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 3 \\ 4 \end{bmatrix}$$

$$W_p(d) = \begin{bmatrix} W_p(1) + W_p(e) \\ W_p(\bullet e) + W_p(e) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 3 \\ 2 \\ 3 \\ 3 \end{bmatrix},$$

$$W_l(d) = \begin{bmatrix} W_p(e) + W_l(1) + W_l(e) \\ W_p(e) + W_l(\bullet e) + W_l(e) \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 4 \\ 3 \end{bmatrix}$$

$$W_p(a) = \begin{bmatrix} W_p(b) + W_p(d) \\ W_p(c) + W_p(d) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 2 \\ 3 \\ 3 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 3 \\ 3 \\ 3 \end{bmatrix} + \begin{bmatrix} 3 \\ 2 \\ 3 \\ 3 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 4 \\ 3 \\ 5 \\ 5 \\ 5 \\ 5 \\ 6 \\ 6 \end{bmatrix},$$

$$W_l(a) = \begin{bmatrix} W_p(d) + W_l(b) + W_l(d) \\ W_p(d) + W_l(c) + W_l(d) \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix}3\\2\\3\\3\end{bmatrix} + \begin{bmatrix}1\\1\\1\\1\end{bmatrix} + \begin{bmatrix}3\\2\\4\\3\end{bmatrix} \\ \begin{bmatrix}3\\2\\3\\3\end{bmatrix} + \begin{bmatrix}2\\3\\3\\4\end{bmatrix} + \begin{bmatrix}3\\2\\4\\3\end{bmatrix} \end{bmatrix} = \begin{bmatrix}7\\5\\8\\7\\8\\7\\10\\10\end{bmatrix}$$

LEMMA 1 *W_p and W_l, with elements sorted in ascending order of magnitudes, are invariant under input negation.*

Proof Negating one or more inputs of an n-variable Boolean function is equivalent to applying the opposite type of Davio's decomposition to every vertex containing the affected variables. Hence, those weights that were computed based on Eq. (5) will be computed by Eq. (6) and vice versa. Since every affected vertex v will be accompanied by a swap of $low(v) \leftrightarrow high(v)$, the weight computed for the FPRM expansion of polarity ω will be the weight of the FPRM expansion computed in the new polarity $\omega' = \phi(\omega)$ and vice versa, where ϕ is the input phase mapping. Hence, negation of any number of input variables result in a permutation of 2^{n-1} pairs of weights w_p (or w_l) in W_p (or W_l). □

LEMMA 2 *W_p and W_l, with elements sorted in ascending order of magnitudes, are invariant under input permutation.*

Proof Consider the partial ETDD shown in Fig. 4. Applying the positive and negative Davio's decomposition around the variable x_j to the children of vertex v, from Eqs. (3) and (4), we have

$$f_{low(v)} = \bar{x}_j f_{xor(low(v))} \oplus f_{high(low(v))}$$
$$= x_j f_{xor(low(v))} \oplus f_{low(low(v))} \quad (7a)$$

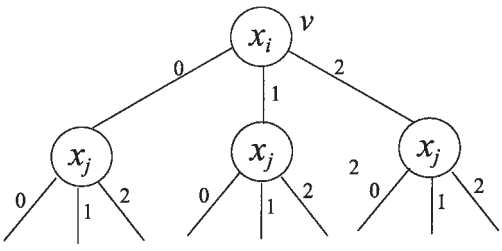

FIGURE 4 A partial ETDD.

$$f_{high(v)} = \bar{x}_j f_{xor(high(v))} \oplus f_{high(high(v))}$$
$$= x_j f_{xor(high(v))} \oplus f_{low(high(v))} \quad (7b)$$

$$f_{xor(v)} = \bar{x}_j f_{xor(xor(v))} \oplus f_{high(xor(v))}$$
$$= x_j f_{xor(xor(v))} \oplus f_{low(xor(v))} \quad (7c)$$

By decomposing the function f_v around the variable x_j, the four expansions with fixed polarities are given as follows:

$$f_v = \bar{x}_i \{\bar{x}_j f_{xor(xor(v))} \oplus f_{high(xor(v))}\} \oplus \bar{x}_j f_{xor(high(v))} \oplus f_{high(high(v))} \quad (8a)$$

$$f_v = \bar{x}_i \{x_j f_{xor(xor(v))} \oplus f_{low(xor(v))}\} \oplus x_j f_{xor(high(v))} \oplus f_{low(high(v))} \quad (8b)$$

$$f_v = x_i \{\bar{x}_j f_{xor(xor(v))} \oplus f_{high(xor(v))}\} \oplus \bar{x}_j f_{xor(low(v))} \oplus f_{high(low(v))} \quad (8c)$$

$$f_v = x_i \{x_j f_{xor(xor(v))} \oplus f_{low(xor(v))}\} \oplus x_j f_{xor(low(v))} \oplus f_{low(low(v))} \quad (8d)$$

Let $\omega = \omega_i \omega_j$. Then, the weights $w_p(\omega)$ are given by:

$$w_p(11) = w_p(f_{xor(xor(v))}) + w_p(f_{high(xor(v))}) + w_p(f_{xor(high(v))}) + w_p(f_{high(high(v))}) \quad (9a)$$

$$w_p(10) = w_p(f_{xor(xor(v))}) + w_p(f_{low(xor(v))}) + w_p(f_{xor(high(v))}) + w_p(f_{low(high(v))}) \quad (9b)$$

$$w_p(01) = w_p(f_{xor(xor(v))}) + w_p(f_{high(xor(v))}) + w_p(f_{xor(low(v))}) + w_p(f_{high(low(v))}) \quad (9c)$$

$$w_p(00) = w_p(f_{xor(xor(v))}) + w_p(f_{low(xor(v))}) + w_p(f_{xor(low(v))}) + w_p(f_{low(low(v))}) \quad (9d)$$

Interchanging variables x_i and x_j in Eqs. (8a)–(8d) results in a new set of weights $w'_p(\omega)$ given by:

$$w'_p(11) = w_p(f_{xor(xor(v))}) + w_p(f_{high(xor(v))}) + w_p(f_{xor(high(v))}) + w_p(f_{high(high(v))}) \quad (10a)$$

$$w'_p(10) = w_p(f_{xor(xor(v))}) + w_p(f_{high(xor(v))}) + w_p(f_{xor(low(v))}) + w_p(f_{high(low(v))}) \quad (10b)$$

$$w'_p(01) = w_p(f_{xor(xor(v))}) + w_p(f_{low(xor(v))}) + w_p(f_{xor(high(v))}) + w_p(f_{low(high(v))}) \quad (10c)$$

$$w'_p(00) = w_p(f_{xor(xor(v))}) + w_p(f_{low(xor(v))}) + w_p(f_{xor(low(v))}) + w_p(f_{low(low(v))}) \quad (10d)$$

TABLE I Weight vectors of three variable NPN-invariant functions

Representative FPRM expansion	Wp	W_l	No. of functions
$x_1 \oplus x_1x_2 \oplus x_2x_3$	{3, 3, 4, 4, 4, 4, 5, 5}	{5, 5, 5, 5, 6, 6, 6, 6}	24
$x_1x_2 \oplus x_3$	{2, 3, 3, 3, 4, 4, 4, 5}	{3, 3, 4, 4, 4, 4, 5, 5}	24
$x_1 \oplus x_2$	{2, 2, 2, 2, 3, 3, 3, 3}	{2, 2, 2, 2, 2, 2, 2, 2}	6
$x_1 \oplus x_2 \oplus x_3$	{3, 3, 3, 3, 4, 4, 4, 4}	{3, 3, 3, 3, 3, 3, 3, 3}	2
$x_1x_2 \oplus x_1x_3 \oplus x_2x_3$	{3, 4, 5, 5, 5, 6, 6, 6}	{6, 6, 8, 8, 8, 8, 8, 8}	8
x_1	{1, 1, 1, 1, 2, 2, 2, 2}	{1, 1, 1, 1, 1, 1, 1, 1}	6
$1 \oplus x_1 \oplus x_2 \oplus x_1x_2x_3$	{4, 4, 4, 4, 4, 5, 5, 7}	{5, 7, 7, 7, 8, 8, 10, 10}	24
$x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2x_3$	{4, 4, 5, 5, 5, 6, 6, 6}	{6, 8, 8, 8, 9, 9, 9, 9}	8
$x_1 \oplus x_1x_2x_3$	{2, 3, 3, 3, 4, 6, 6, 6}	{4, 6, 6, 6, 7, 9, 9, 11}	24
$x_1 \oplus x_2 \oplus x_1x_2x_3$	{3, 4, 5, 5, 5, 5, 6, 6}	{5, 7, 7, 7, 8, 8, 10, 10}	24
$1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2x_3$	{5, 5, 5, 5, 5, 6, 6, 6}	{6, 8, 8, 8, 9, 9, 9, 9}	8
$1 \oplus x_1 \oplus x_1x_2x_3$	{3, 3, 4, 4, 4, 5, 5, 7}	{4, 6, 6, 6, 7, 9, 9, 11}	24
x_1x_2	{1, 1, 2, 2, 2, 2, 4, 4}	{2, 2, 3, 3, 3, 3, 4, 4}	12
$x_1x_2 \oplus x_1x_3$	{2, 2, 3, 3, 4, 4, 6, 6}	{4, 4, 5, 5, 6, 6, 7, 7}	12
$x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3$	{4, 4, 4, 4, 4, 4, 7, 7}	{7, 7, 7, 7, 7, 7, 9, 9}	4
$1 \oplus x_1x_2$	{2, 2, 3, 3, 3, 3, 3, 3}	{2, 2, 3, 3, 3, 3, 4, 4}	12
$1 \oplus x_1x_2 \oplus x_1x_3$	{3, 3, 4, 4, 5, 5, 5, 5}	{4, 4, 5, 5, 6, 6, 7, 7}	12
$1 \oplus x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3$	{5, 5, 5, 5, 5, 5, 6, 6}	{7, 7, 7, 7, 7, 7, 9, 9}	4
$x_1x_2x_3$	{1, 2, 2, 2, 4, 4, 4, 8}	{3, 5, 5, 5, 8, 8, 8, 12}	8
$1 \oplus x_1x_2x_3$	{2, 3, 3, 3, 5, 5, 5, 7}	{3, 5, 5, 5, 8, 8, 8, 12}	8
1	{1, 1, 1, 1, 1, 1, 1, 1}	{0, 0, 0, 0, 0, 0, 0, 0}	1
0	{0, 0, 0, 0, 0, 0, 0, 0}	{0, 0, 0, 0, 0, 0, 0, 0}	1

It can be observed that $w_p(\omega_i\omega_j) = w'_p(\omega_j\omega_i)$. Similar proof can also be derived for $w_l(\omega_i\omega_j) = w'_l(\omega_j\omega_i)$. Since the variables x_i and x_j are arbitrary, there exists a bijection between W' and W for any permutation of input variables. □

LEMMA 3 *Complementing a function does not alter its w_l, but either increments or decrements its w_p by one, for all polarity ω.*

Proof Trival since $\overline{A^{\omega}} = 1 \oplus A^{\omega}$ for any ω. □

Table I shows the weight vectors W_p and W_l with their elements arranged in ascending order of magnitudes for 22 NP-invariant classes for all three variable Boolean functions. The first column gives the representative FPRM expansion of each class and the last column gives the total number of functions in each class. The last two rows of Table I correspond to the trival functions $F = 1$ and $F = 0$.

SYMBOLIC METHOD FOR COMPUTATION FOR REED–MULLER WEIGHT VECTORS

The method of calculating the FPRM weights, as described by the algorithm in Fig. 2 and illustrated by Example 1, although simple and fast, suffers from one main disadvantage of high storage requirements. This is due to the fact that all the vector additions are performed over the subvectors represented by the subtrees in the ETDD. The subvector required for holding the partial weights is embedded in the structure of every vertex, and the memory allocated for such an array increases exponentially as the index of the vertex decreases.

Symbolic methods in the form of Algebraic Decision Diagrams have been proposed for the computation of various discrete transforms [1,15]. Algebraic Decision Diagrams (ADDs), also known as Multiple Terminal Binary Decision Diagrams (MTBDDs), are Binary Decision Diagrams with multiple valued vertices. By exploiting the recurrence relation of the transforms, such methods are capable of processing in-place Fast Fourier Transform like algorithms [1,16] for complex functions with large number of input variables efficiently. In order to overcome the problem of dimensionality, the intermediate vector addition results and the derivation of the final weight vectors will be embodied in the ADD data structure similar to the approach used in Ref. [15]. The operations defined by Eqs. (3) and (4) can be executed recursively by the well known *apply* operation [12] on the ADD, beginning from the root. The algorithm for the calculation of the entire weight vector, W_p by symbolic manipulation on the ADD is shown in Fig. 5.

In Fig. 5, the input argument v is a vertex of the ETDD representing the Boolean function, and the variables *v_low*, *v_high*, *v_xor*, *u_low*, *u_high* and *result* are ADD vertices. The recursive algorithm begins with a depth first transversal of the ETDD of the function until the terminal vertices. The resulting ADD is constructed in a bottom up manner by forming an ADD representing the weight subvector of each ETDD vertex visited. A unique table is established to maintain isomorphic ADD vertices of the resulting graph. The procedure **find_terminal** with input *value*(v) finds an ADD terminal vertex with the value, *value*(v) from the unique table if it already exists or creates a new terminal vertex otherwise. Likewise, the procedure **find_nonterminal** searches the unique table for an existing non-terminal vertex with the given index and children or generates a new one if the vertex with the input arguments has not yet been created. The routine **apply_add** returns the ADD of the addition of two ADD

```
ADD_Wp(v)
{
    if (index(v) > n) return find_terminal(value(v));
    v_low = ADD_Wp(low(v));
    v_high = ADD_Wp(high(v));
    v_xor = ADD_Wp(xor(v));
    u_low = apply_add(v_low, v_xor);
    u_high = apply_add(v_high, v_xor);
    result = find_nonterminal(index(v), u_low, u_high);
    return result;
}
```

FIGURE 5 Symbolic calculation of weight vector.

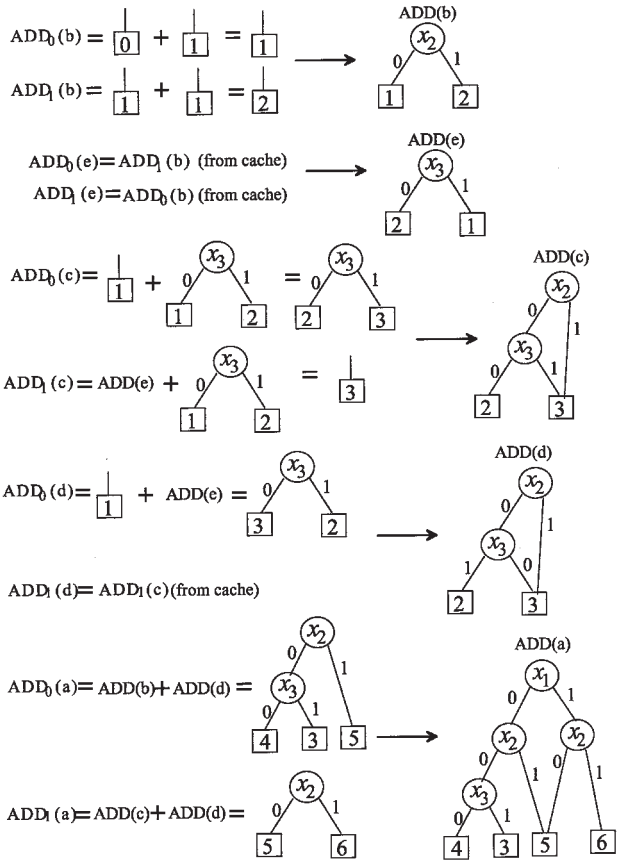

FIGURE 6 Generation of weight vector in ADD presentation.

operands in a recursive way similar to the standard *apply* operation in BDD [12]. Although not indicated in the algorithm, a computational cache is used to speed up the computation by eliminating multiple operations on the same operands. Before carrying out the *apply* operation, the computational cache is searched for previously computed results based on the supplied operator and operands. The latest computational results will also be inserted into the cache. Besides the *apply* operation, the calculated ADD representation of the subvector embedded in each vertex of the ETDD will also be cached. The usage of the computational cache significantly improves the processing speed for ETDD of functions with a large number of shared vertices.

For the calculation of weight vector W_l, the same algorithm can be used by embracing in the recursion an *apply* operation on the result retuned by the call to **ADD_W_p**($xor(v)$).

Example 2 Figure 6 shows the trace of computation of the weight vector W_p from the ETDD of Fig. 3. If v is the ETDD vertex, $\mathrm{ADD}_1(v)$ and $\mathrm{ADD}_0(v)$ are the ADD representations corresponding to the upper and lower halves of the weight subvectors calculated for v in Example 2. The concatenation of subvectors is formed by the composition of **if** ADD(v) **then** $\mathrm{ADD}_1(v)$ **else** $\mathrm{ADD}_0(v)$ recursively. In Fig. 6, the overheads saved by results returned directly from computational cache are also shown. The resulting weight vector W_p is stored as an ADD of four non terminal vertices and four terminal vertices. The number of product terms for any FPRM polarity can be read from the ADD terminal vertices by following the edges with values corresponding to the polarity bits.

BOOLEAN MATCHING FILTERS

Let f be a function of a subcircuit and $G = \{g_1, g_2, \ldots, g_n\}$ be the set of library functions of the target technology. Then the basic Boolean matching problem is to find which library function g_i can be transformed to f under appropriate NPN operations. Two functions, $f = \{x_1, x_2, \ldots, x_n\}$ and $g = \{y_1, y_2, \ldots, y_n\}$ are NPN equivalent (written as $f \equiv g$) if there exist transformations, π, ϕ and φ such that $g(Y) = \varphi \circ f(\pi \circ \phi \circ X)$. The transformation $\phi \in \mathbf{B}^n$ is a phase assignment to input variables, and $\phi : \{x_1, x_2, \ldots, x_n\} \rightarrow \{\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n\}$ where $\dot{x}_i = \bar{x}_i$ or x_i. $\pi \in \mathbf{S}^n$ is a permutation of input variables where $\pi = \{x_1, x_2, \ldots, x_n\} \rightarrow \{y_1, y_2, \ldots, y_n\}$ and $y_j = \pi(x_i)$ where $i = j$ or $i \neq j$ and $1 \leq i, j \leq n$. $\varphi \in \mathbf{B}$ is a phase assignment to the function's output, and $\varphi : f \rightarrow f$ or $\bar{f}$. A direct enumeration of all transformations on a function f to determine a match involves $2^{n+1}\, n!$ operations. To reduce the search space, matching filters exploiting the necessary conditions for NPN equivalent of two functions are used. The method employs a set of characteristic values for the function and its variables, called the signature, to distinguish itself from the non NPN equivalent functions and to identify the variable correspondence between the matched functions. The signatures should be computed at reasonably low time and space complexity. Given the signatures, $S(f)$ and $S(g)$ of f and g, respectively, $S(f) \neq S(g)$ implies that no π, ϕ and φ could match f to g. Aliasing occurs when $S(f) = S(g)$ but the two functions are not matched. A good matching filter will not only minimize the aliasing occurrences, but it will also restrict the domains of π and ϕ mappings required for equivalent verification when $S(f) = S(g)$.

From the NPN invariant properties of the weight vectors presented in the previous section, it is evident that the weight vector W_t, with appropriately sorted elements, constitutes a zero aliasing filter. However, the 2^n elements in the weight vector require excessive memory space for large library. It is possible to avoid the dimensionality problem by deriving a set of low aliasing filters from selected components of the weight vectors. For each function f, the signature is divided into two levels: a functional level signature, $S(f)$ is computed directly from the function itself, and a variable level signature, $S(f, \dot{x}_i)$, is computed from the $2n$ cofactors, $f_{\bar{x}_i}$ and $f_{x_i} (i = 1, 2, \ldots, n)$. The variable level signature is useful in detecting the input phase transformation. If $f \equiv_{np} g$ and $\pi(\phi(y_j)) = \bar{x}_i$, then $S(f, \bar{x}_i) = S(g, y_j)$. We propose that the signatures $S(f)$ and $S(f, \dot{x}_i)$ be composed from any or a hybrid combination of the following elements:

(1) $\min\limits_{0 \le \omega < 2^n}(w_p)$ and $\left|\min\limits_{0 \le \omega < 2^n}(w_p)\right|$

(2) $\max\limits_{0 \le \omega < 2^n}(w_p)$ and $\left|\max\limits_{0 \le \omega < 2^n}(w_p)\right|$

(3) $\min\limits_{0 \le \omega < 2^n}(w_l)$ and $\left|\min\limits_{0 \le \omega < 2^n}(w_l)\right|$

(4) $\max\limits_{0 \le \omega < 2^n}(w_l)$ and $\left|\max\limits_{0 \le \omega < 2^n}(w_l)\right|$

In the above equations, the symbol $|w|$ means the number of times integer w repeats itself in the given vector. Since W_p with appropriately sorted elements is only NP equivalent, when the signatures are to be composed from Eqs. (1) and (2), the signatures will be computed based on both $\bar{f}$ and f if $|f| = |\bar{f}|$. Otherwise, the signatures can be calculated from either f or $\bar{f}$ if $|f| < 2^{n-1}$ or $|\bar{f}| < 2^{n-1}$, accordingly. Complementing the function can be achieved easily with the use of the output negation attribute. The basis of this output phase assignment φ is explained as follows:

Let $\vartheta = \{\vartheta_1, \vartheta_2, \ldots, \vartheta_m\}$ be the set of NPN equivalent classes formed by all n variable Boolean functions, and $|\vartheta_i|$ denote the number of functions in the class ϑ_i. For any class $\vartheta_i \in \vartheta$, we can further divide it into three NP equivalent subclasses ϑ_{i0}, ϑ_{i1} and ϑ_{i2}, where $\vartheta_{i0} = \{f \in F \| f| < 2^{n-1}\}$, $\vartheta_{i1} = \{f \in F \| f| > 2^{n-1}\}$ and $\vartheta_{i2} = \{f \in F \| f| = |\bar{f}|\}$. For every function $f \in \vartheta_{i1}, \bar{f} \in \vartheta_{i0}$ and vice versa. However, for every function $f \in \vartheta_{i2}, \bar{f} \in \vartheta_{i2}$.

Example 3 Consider the hybrid type 1 and 2 functional level signature $S(f) = \left[\min\limits_{0 \le \omega < 2^n}(w_p) \left|\min\limits_{0 \le \omega < 2^n}(w_p)\right| \max\limits_{0 \le \omega < 2^n}(w_p) \left|\max\limits_{0 \le \omega < 2^n}(w_p)\right|\right]$. From Example 1, for the function $f = \Sigma m(2, 3, 4, 5, 6)$, we have $S(f) = [3162]$. The variable level signature $S(f, \dot{x}_i)$ can be calculated through the same ETDD by bypassing the vertices containing the variable x_i through its low child if $\dot{x}_i = \bar{x}_i$ and high child if $\dot{x}_i = x_i$. The computation of W_p for $S(f, x_2)$ is illustrated as follows:

$$W_p(a) = \begin{bmatrix} W_p(1) + W_p(\bullet e) \\ W_p(e) + W_p(\bullet e) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\ \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 3 \\ 3 \\ 3 \end{bmatrix}$$

Hence, $S(f, x_2) = [2133]$.

The variable level signatures are calculated and summarized as follows:

$$S(f, \bar{x}_1) = [1222], S(f, x_1) = [2133],$$

$$S(f, \bar{x}_2) = [1222], S(f, x_2) = [2133],$$

$$S(f, \bar{x}_3) = [2133], S(f, x_3) = [2232].$$

Both types of variable level signatures have successfully distinguished the variable x_3 from the other variables, but they fail to distinguish the two variables x_1 and x_2. In fact, x_1 and x_2 form a symmetric set of f.

Example 4 Consider another three variable function $g(Y) = \Sigma m(3, 4, 7)$. Since $|g| < 4$, we will compute the signatures based on $\bar{g} = \Sigma m(0, 1, 2, 5, 6)$. The functional level signature $S(\bar{g}) = [3162]$. Since $S(\bar{g}) = S(f)$, $\bar{g}$ may be matched to f. The variable level signatures of g are calculated: $S(\bar{g}, \bar{y}_1) = [2133]$, $S(\bar{g}, y_1) = [2232]$, $S(\bar{g}, \bar{y}_2) = [2133]$, $S(\bar{g}, y_2) = [1222]$, $S(\bar{g}, \bar{y}_3) = [2133]$ and $S(\bar{g}, y_3) = [1222]$. Comparing with the variable signatures of f, the following input correspondence between f and $\bar{g}$ can be established: $\pi = (x_3) = y_1, \pi(\phi\{x_1, x_2\}) = \{\bar{y}_2, \bar{y}_3\}$. In fact, f and g are equivalent under the above input and output transformations.

Besides the minimal and maximal weights, we can introduce the second, third, fourth, etc. smallest weights with their respective cardinalities as additional signatures. When constructing the library, these extended sets of signatures can be generated adaptively to the aliasing variables for those NPN classes with identical signatures. Thus, instead of allocating constant storage to every NPN representative function in the library, some NPN representatives may require bigger number of signatures and memory than others to effectively reduce the aliasing probability.

The above signatures are directly related to the selected row sums of the polarity coefficient matrix. We will also examine the column sums of the polarity coefficient matrix to derive another computationally efficient signature. To simplify the discussion, we consider the general form of polarity coefficient matrix $PC(F)$ of an

$$
\begin{bmatrix}
m_0 & m_0\oplus m_1 & m_0\oplus m_2 & \oplus m(0,1,2,3) & m_0\oplus m_4 & \oplus m(0,1,4,5) & \oplus m(0,2,4,6) & \bigoplus_{i=0}^{7} m_i \\
m_1 & m_0\oplus m_1 & m_1\oplus m_3 & \oplus m(0,1,2,3) & m_1\oplus m_5 & \oplus m(0,1,4,5) & \oplus m(1,3,5,7) & \bigoplus_{i=0}^{7} m_i \\
m_2 & m_2\oplus m_3 & m_0\oplus m_2 & \oplus m(0,1,2,3) & m_2\oplus m_6 & \oplus m(2,3,6,7) & \oplus m(0,2,4,6) & \bigoplus_{i=0}^{7} m_i \\
m_3 & m_2\oplus m_3 & m_1\oplus m_3 & \oplus m(0,1,2,3) & m_3\oplus m_7 & \oplus m(2,3,6,7) & \oplus m(1,3,5,7) & \bigoplus_{i=0}^{7} m_i \\
m_4 & m_4\oplus m_5 & m_4\oplus m_6 & \oplus m(4,5,6,7) & m_0\oplus m_4 & \oplus m(0,1,4,5) & \oplus m(0,2,4,6) & \bigoplus_{i=0}^{7} m_i \\
m_5 & m_4\oplus m_5 & m_5\oplus m_7 & \oplus m(4,5,6,7) & m_1\oplus m_5 & \oplus m(0,1,4,5) & \oplus m(1,3,5,7) & \bigoplus_{i=0}^{7} m_i \\
m_6 & m_6\oplus m_7 & m_4\oplus m_6 & \oplus m(4,5,6,7) & m_2\oplus m_6 & \oplus m(2,3,6,7) & \oplus m(0,2,4,6) & \bigoplus_{i=0}^{7} m_i \\
m_7 & m_6\oplus m_7 & m_5\oplus m_7 & \oplus m(4,5,6,7) & m_3\oplus m_7 & \oplus m(2,3,6,7) & \oplus m(1,3,5,7) & \bigoplus_{i=0}^{7} m_i
\end{bmatrix}
$$

pi-term	1	x_1	x_2	x_2x_1	x_3	x_3x_1	x_3x_2	$x_3x_2x_1$

FIGURE 7 Polarity coefficient matrix of three variable Boolean function.

arbitrary three variable Boolean function F expressed in terms of its minterms. The representation is shown in Fig. 7. It is noticed that each column $j(j = j_3j_2j_1)$ of $PC(F)$ represents a particular pi-term of the FPRM expansion, $\dot{x}_3^{j_3}\dot{x}_2^{j_2}\dot{x}_1^{j_1}\left(\dot{x}_i^1 = \dot{x}_i \text{ and } \dot{x}_i^0 = 1\right)$. In general, the j-th column sum $(j = 0, 1, \ldots, 2^n - 1)$ of the polarity coefficient matrix provides the total number of literals present in the j-th pi-term of all 2^n FPRM expansions. It should be noted that for $j = 1, \ldots, 2^n - 1$, the j-th column sum is divisible by 2. For an n-variable Boolean function, the $n + 1$ column sum for the constant and single literal pi-term are of special interest as Boolean matching filter.

Let $S_{C0}(f)$ be the sum of the first column (i.e. column number 0) of $PC(f)$, and $S_C(f)$ be the set $\{S_{C1}(f), S_{C2}(f), \ldots, S_{Cn}(f)\}$ where $S_{Ci}(f), i = 1, 2, \ldots, n$, is equal to half the sum of the j-th column $(j = 2^{i-1})$ of $PC(f)$. Then,

$$S_{C0}(f) = |f|, \text{ and } S_{Ci}(f) = \sum_{|q-p|=2^{i-1}} (m_p \oplus m_q). \tag{11}$$

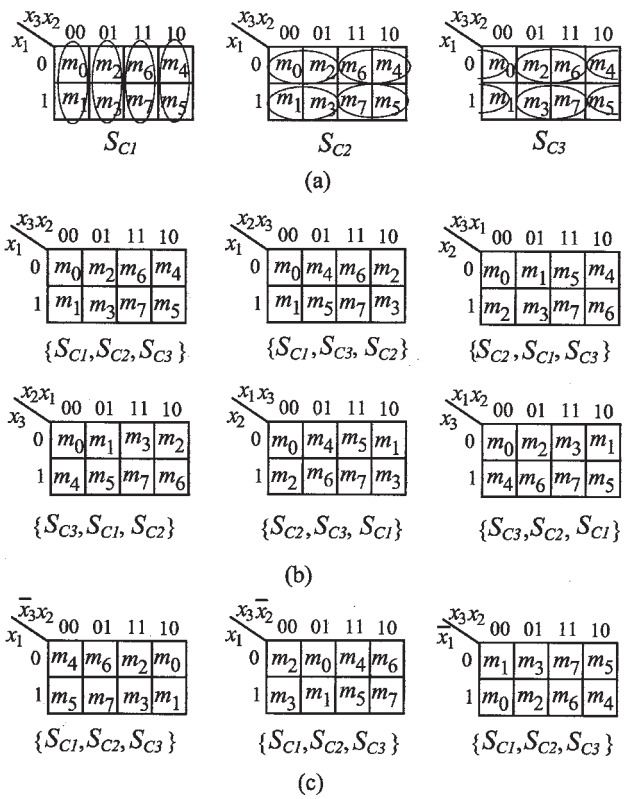

FIGURE 8 Signature S_c: (a) Interpretation of S_{C1}, S_{C2} and S_{C3}. (b) Effect on input permutation. (c) Effect on negation of an input variable.

THEOREM 1 *If $f \equiv_{npn} g$, then $S_{C0}(f) = S_{C0}(g)$ or $2^n - S_{C0}(g)$, and $S_c(f) = S_c(g)$.*

Proof It is trivial to prove that if $f \equiv_{npn} g, |f| = |g|$ if $f \equiv_{np} g$ or $2^n - |g|$ if $f \equiv_{np} \bar{g}$. We will consider the effects of input negation, input permutation and output negation independently on S_{Ci} for $i = 1, 2, \ldots, n$. Let $\{m_p, m_q\}$ be the pair of adjacent minterms different in the variable x_i, where $i = 1, 2, \ldots, n$. Negating the variable x_i makes $m_p \leftrightarrow m_q$. Hence, S_{Ci} remains unchanged. Under the permutation of input variables, x_i and x_j, any pair of minterms $\{m_p, m_q\}$ with $q - p = 2^{i-1}$ will map to $\{m_{p'}, m_{q'}\}$ with $q' - p' = 2^{j-1}$ and vice versa. Hence, if $x_i \leftrightarrow x_j$, $\sum_{|q-p|=2^{i-1}} (m_p \oplus m_q) \leftrightarrow \sum_{|q'-p'|=2^{j-1}} (m_{p'} \oplus m_{q'})$. If $f \equiv_p g$, then $f(X) = g(\pi(X))$ and $S_{Ci}(f) = S_{Cj}(g)$, where $x_i = \pi(x_j)$. Negating the output of a function complements all its minterms. Since $\overline{m_p} \oplus \overline{m_q} = m_p \oplus m_q$, $S_{Ci}(i = 1, 2, \ldots, n)$ is also invariant under output negation. □

Figure 8 shows the effect of input negation and permutation on $\{S_{C1}(F), S_{C2}(F), S_{C3}(F)\}$ for a three variable function F. It should be noted that, while the correspondence of variables under input permutation can be identified by the order of the unique elements of S_C, the negated variable can not be identified from the order or values of the elements of S_C. The input negation mapping can be obtained by means of variable level signature

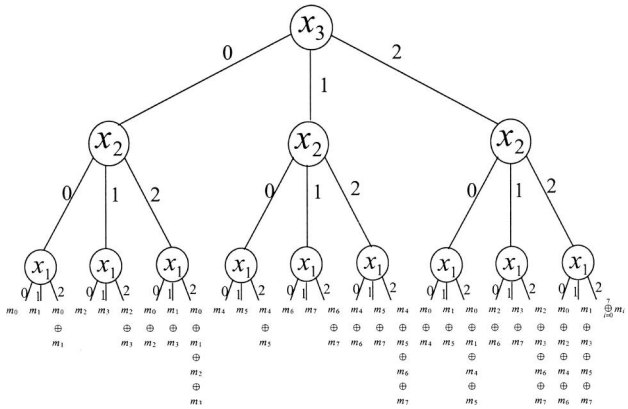

FIGURE 9 EXOR Ternary Decision Tree of three variable functions

similar to that presented in Symbolic Method for Computation of Reed–Muller Weight Vectors. The signature $S_C(f)$, when applied to the $2n$ cofactors of f, forms the variable level signature $S_C(f, \dot{x}_i)$ for $i = 1, 2, \ldots, n$.

Consider the ternary decision tree of a three variable function F shown in Fig. 9. The values of $\{S_{C1}(F), S_{C2}(F), S_{C3}(F)\}$ can be obtained as follows:

$$S_{C1}(F) = value(xor(e)) + value(xor(f)) + value(xor(h)) + value(xor(i))$$

$$S_{C2}(F) = value(low(g)) + value(high(g)) + value(low(j)) + value(high(j))$$

$$S_{C3}(F) = value(low(k)) + value(high(k)) + value(low(l)) + value(high(l))$$

In general, the signatures, $S_{C0}(g)$ and $S_C(f) = \{S_{C1}(f), S_{C2}(f), \ldots, S_{Cn}(f)\}$ can be computed in a single depth-first traversal of the ETDD. For any non terminal vertex v, having x_i as its variable, if $xor(v)$ is a terminal vertex, its value will be added directly to $S_{Ci}(F)$. When $xor(v)$ is a non terminal vertex, $S_{Ci}(F) = value(low(xor(v))) + value(high(xor(v)))$. The value of $S_{C0}(F)$ can be obtained by recursively adding the values of the low and high children from the leaves to the root. The algorithm **csum** is shown in Fig. 10. Whenever an *xor* child for some node has been visited, the values of the signatures will be calculated from the node's *low* and *high* children. In order to calculate all the elements of $S_C(F)$ in a single traversal, the variable *mark* is used as an input argument to keep track of the traversal of *xor* child. The variables $w0$, $w1$, and $w2$ account for the weights contributed by the missing vertices due to the ETDD reduction rules. The variables *mark* and S_{Ci} for $i = 0, 1, \ldots, n$ are initialized to zero.

```
csum(v, mark)
{
    if (index(v) > n) return value(v);
    i = index of the variable of v;
    w0 = index(low(v)) − index(v) − 1;
    w1 = index(high(v)) − index(v) − 1;
    if (mark) return 2^w0 × csum(low(v), mark) + 2^w1 × csum(high(v), mark);
    else {
            w2 = index(xor(v)) − index(v) − 1;
            S_Ci = S_Ci + 2^w2 × csum(xor(v), 1);
    }
    S_C0 = S_C0 + 2^w0 × csum(low(v), 0) + 2^w1 × csum(high(v), 0);
}
```

FIGURE 10 Algorithm for the computation of S_C.

Example 5 Consider the ETDD in Fig. 3. The trace of the execution of **csum**$(a, 0)$ is given as follows:

At vertex a: $S_{C1} = \mathbf{csum}(d,1) = 2 \times 1 + \mathbf{csum}(\bullet e,1) = 2 + 1 + 0 = 3.$
$S_{C0} = \mathbf{csum}(b,0) + \mathbf{csum}(c,0).$

At vertex b: $S_{C2} = 2 \times 1 = 2.$ $S_{C0} = 2 \times 0 + 2 \times 1 + \mathbf{csum}(c, 0) = 2 + \mathbf{csum}(c, 0).$

At vertex c: $S_{C2} = 2 + \mathbf{csum}(\bullet e, 1) = 2 + 1 + 0 = 3.$
$S_{C0} = 2 + 2 \times 1 + \mathbf{csum}(e, 0) = 4 + \mathbf{csum}(e, 0).$

At vertex e: $S_{C3} = 1.$ $S_{C0} = 4 + 1 + 0 = 5.$

Hence, $S_{C0}(f) = 5$ and $S_C(f) = \{3, 3, 1\}$. The results of the calculation of variable level signatures $S_C(f, \dot{x}_i)$ are given as follows:

$S_{C0}(f, \bar{x}_1) = 2,\ S_{C2}(f, \bar{x}_1) = 2,\ S_{C3}(f, \bar{x}_1) = 0.$
$S_{C0}(f, x_1) = 3,\ S_{C2}(f, x_1) = 1,\ S_{C3}(f, x_1) = 1.$
$S_{C0}(f, \bar{x}_2) = 2,\ S_{C1}(f, \bar{x}_2) = 2,\ S_{C3}(f, \bar{x}_2) = 0.$
$S_{C0}(f, x_2) = 3,\ S_{C1}(f, x_2) = 1,\ S_{C3}(f, x_2) = 1.$
$S_{C0}(f, \bar{x}_3) = 3,\ S_{C1}(f, \bar{x}_3) = 1,\ S_{C2}(f, \bar{x}_3) = 1.$
$S_{C0}(f, x_3) = 2,\ S_{C1}(f, x_3) = 2,\ S_{C2}(f, x_3) = 2.$

Consider another three variable function $g(Y) = \Sigma m(3, 4, 7)$ from Example 4. Its functional level signatures are $S_{C0}(g) = 3$ and $S_C(g) = \{1, 3, 3\}$. In addition, we have $S_{C0}(g, \bar{y}_1) = S_{C2}(g, \bar{y}_1) = S_{C3}(g, \bar{y}_1) = 1.$ $S_{C0}(g, y_1) = S_{C2}(g, y_1) = S_{C3}(g, y_1) = 2.$ $S_{C0}(g, \bar{y}_2) = S_{C1}(g, \bar{y}_2) = S_{C3}(g, \bar{y}_2) = 1.$ $S_{C0}(g, y_2) = S_{C3}(g, y_2) = 2,$ $S_{C1}(g, y_2) = 0.$ $S_{C0}(g, \bar{y}_3) = S_{C1}(g, \bar{y}_3) = S_{C2}(g, \bar{y}_3) = 1.$ $S_{C0}(g, y_3) = S_{C2}(g, y_3) = 2,$ $S_{C1}(g, y_3) = 0.$ We can observe the consistency between the functional and variable level signatures in terms of correspondence between output negation and input permutation. For instance, the following relations exist between the variables x_3 and y_1. $S_{C0}(g) = 8 - S_{C0}(f),$ $S_{C0}(g, y_1) = 4 - S_{C0}(f, x_3),$ $S_{C0}(g, \bar{y}_1) = 4 - S_{C0}(f, \bar{x}_3),$ $S_{C3}(g, y_2) = S_{C1}(f, \bar{x}_2),$ $S_{C1}(g, y_3) = S_{C1}(g, y_3) = S_{C3}(g, \bar{x}_1) = S_{C3}(g, \bar{x}_2)$, etc. The same conclusion as that in Example 4 about the equivalence of f and g can be established.

STRING REPRESENTATION OF SIGNATURES

Bit strings or character strings can be used directly as a key for searching in a hash table [19]. Moreover they can

TABLE II Effectiveness of Boolean matching filters

Type	NP3			NPN3			NP4			NP4		
	η	u	A	η	u	A	η	u	A	η	u	A
F1	22	1	0	14	1	0	239	0.5945	47.94	163	0.7342	16.87
F2	22	1	0	14	1	0	402	1	0	222	1	0
F3	–	–	–	14	1	0	–	–	–	174	0.7838	14.82
F4	–	–	–	14	1	0	–	–	–	222	1	0
F5	22	1	0	14	1	0	391	0.9726	1.28	216	0.9730	0.68

be further compressed using standard compression techniques such as run length coding [17]. Instead of comparing each element of the vector representation of the characteristic signature S, we exploit the lexicographic order of character strings for fast comparison. The process of transforming the characteristic signature from vector into string representation consists of partial sorting and concatenation.

Let a|b denotes the concatenation of symbols a and b. Let $\tau : \mathbf{S}^n \rightarrow \mathbf{S}$ represents the mapping of a vector of n symbols into a single symbol string by concatenation. $\tau(S) = s_1|s_2|\ldots|s_i|s_{i+1}|\ldots|s_n$, where s_i is the i-th element of the vector S. Let $\sigma : \mathbf{S}^n \rightarrow \mathbf{S}^n$ be the reordering of n symbol strings based on the lexicographic order, $<_{\text{lex}}$ in the symbol space. If S is an array of character strings and $s_i <_{\text{lex}} s_j$ for any $s_i, s_j \in S$, then s_i will appear before s_j in $\sigma(S)$. For simplicity, the symbols involved in the mappings τ and σ are assumed to be characters. Non character symbols will be converted to characters before the mapping.

Based on the above definitions, the characteristic signature of the Boolean matching filter based on the maximum and minimum FPRM weights and the number of optimal and worst case expansions can be compressed into a single character string C as follows:

$$C(f) = \tau(\tau(S(f)), \sigma(\tau(S(f, \bar{x_1})), \tau(S(f, x_1)), \cdots, \tau(S(f, \bar{x_n})), \tau(S(f, x_n)))) \quad (12)$$

For characteristic signature based on the column sum of the polarity coefficient matrix, the string representation can be obtained by the above transformation with $\tau(S(f))$ replaced by $\tau(S_{C0}(f), \sigma(S_{C1}(f), S_{C2}(f), \cdots, S_{Cn}(f)))$ and $S(f, \dot{x_i})$ by $\tau(S_{C0}(f, \dot{x_i}), \sigma(S_{C1}(f, \dot{x_i}), S_{C2}(f, \dot{x_i}), \cdots, S_{Cn}(f, \dot{x_i})))$. In addition, $S_{C0}(f)$ will be replaced by $1 - S_{C0}(f)$ if it is greater than 2^{n-1}, and $S_{C0}(f, \dot{x_i})$ replaced by $1 - S_{C0}(f, \dot{x_i})$ if it is greater than 2^{n-2}.

It could be easily verified that the transformation given by Eq. (12) will not alter the effectiveness of the filter in distinguishing non equivalent functions.

Example 6 Based on Eq. (12), the character string representation of the filter for f from Example 3 is given by: $C(f) = \tau(3162, \sigma(1222, 2133, 1222, 2133, 2133, 2232)) = \tau(3162, 1222, 1222, 2133, 2133, 2133, 2232) =$ 3162122212222133213321332232. The signature can be reduced to a 18 byte character string like 3162(1222)2(2133)32232 by identifying repetitive patterns.

If the integer values, z of the signature is converted to a binary word with word length equal to $\lceil \log_2 z \rceil$, where $\lceil a \rceil$ denotes the smallest integer greater than a, the signature $C(f) =$ 111110101101010110101010111110111111011111110101110. This binary bit stream requires only seven bytes of storage. The binary string representation of the signature with long runs of consecutive "1"s and "0"s can be further compressed by simple run length coding to reduce the memory requirement. Run length code will reduce the storage requirement of this binary string to six bytes.

EXPERIMENTAL RESULTS

An NP (or NPN) filter partitions the space of all Boolean functions of n variables into η equivalent signature classes. In Ref. [4], the figure of merit, $u =$ number of distinct signatures in the library, η/total number of basis functions in the library is used for measuring the effectiveness of a signature scheme. As the number of Boolean functions belonging to each equivalent signature classes varies, it would be interesting to examine the probability of aliasing from experimental results based on the cardinality of the colliding classes. The percentage aliasing, A is calculated by:

$$A = \frac{\sum_{i=1}^{p} |A_i|}{2^{2^n}} \times 100\%$$

where n is the number of variables, p is the total number of aliasing classes and $|A_i|$ is the number of functions in the i-th aliasing class A_i. The lower the value of A of a NP-equivalent filter, the higher the likelihood that two arbitary chosen non NP-equivalent functions will end out with different signatures.

The algorithms for the computations of various signatures represented as integer strings have been implemented in C language. The effectiveness of the filters proposed in Boolean Matching Filters have been evaluated based on the NP and NPN equivalent class representative functions from Ref. [10]. The results for three and four variable functions are given in Table II. Due to the huge number of NP and NPN equivalent classes,

representative functions of five variable and above are not evaluated. It should be noticed that in practice the functions of few variables have to be matched in Boolean mapping [6,15]. In Table II, the various types of filters used are given in the first column. "F1" represents the functional level signature of type 1 and 2, in Symbolic Method for Computation of Reed–Muller Weight Vectors, i.e. of the form $\{w_{p\max}, |w_{p\max}|, w_{p\min}, |w_{p\min}|\}$. "F2" represents the combined functional and cofactor signature of the same type as "F1". "F3" represents the functional level signature of type 3 and 4 in Symbolic Method for Computation of Reed–Muller Weight Vectors, i.e. of the form $\{w_{l\max}, |w_{l\max}|, w_{l\min}, |w_{l\min}|\}$. "F4" represents the combined functional and cofactor signature of the same type as "F3". "F5" is the combined functional and cofactor signature based on column weights of the polarity coefficient matrix. The columns labeled "NP3" and "NPN3" list the results obtained from 22 NP equivalent classes and 14 NPN equivalent classes for functions of three variables, respectively. The column labeled "NP4" and "NPN4" list the results obtained from 402 NP equivalent and 222 NPN equivalent classes for functions of four variables, respectively. The results present the number of unique signatures (η), the figure of merit (u) and the percentage aliasing (A). As filters based on Reed–Muller literals vector have been proven to be ineffective in distinguishing two different NP equivalent classes which are output negation equivalent, the results of F3 and F4 for NP equivalent functions are irrelevant. All the proposed filters have achieved zero aliasing for the basis functions of three variables. The results show that functional level signature alone (F1, F3) based on either the FPRM literals or products is capable of distinguishing all NPN and NP (for F1) equivalent classes of three variables. For four variable functions, only filters F2 and F4 have zero aliasing for NPN equivalent verification, and F2 has zero aliasing for NP equivalent verification. Filter F5 has relatively low aliasing between 1.28 and 0.68% for NP and NPN equivalent functions, respectively. Our signature schemes F2 and F4 have outperformed the signature schemes SIG1 and SIG3 in Ref. [4] and are as good as SIG2 of [4] for four variable functions. However, all our proposed schemes require linear time and space complexity while SIG2 requires exponential space complexity [4]. It should be mentioned that although the Boolean function classification method in Ref. [18] also uses FPRM forms, the parameters extracted from the FPRM expansions are different. The method in Ref. [18] is based on the number of cubes containing certain variables from a single FPRM form. The polarity of the FPRM expansion is first chosen based on the balanced variable criterion. The signature scheme in Ref. [18] is more complex as it considers the number of cubes of each length at the functional level, the column sums of the variable inclusion count matrix, and the column sums of the incidence matrix.

Table III shows the results of the algorithm **ADD_Wp** executed on a HP C180 workstation. For each of the two-level examples of MCNC benchmark functions, its execution time in second is given in the column labeled "cpu". The sizes of the ETDDs of the functions and the sizes of the ADDs of the calculated weight vectors are given in column labeled "ETDD" and "ADD", respectively. The column labeled "#in" shows the number of input variables. As the implementation considered only single output functions, the output of any multiple output function is randomly selected and the selected output is indicated by number following the underscore. The results provide a reasonable assessment of the speed of computation of the filters F1 and F2 for functions with higher number of input variables.

TABLE III Experimental Results for **ADD_wp**

MCNC	#in	Cpu	ETDD	ADD
9sym	9	0.02	72	46
sao2_3	10	0.02	173	800
cordic_1	23	0.10	129	8466
cordic_2	23	0.10	85	5686
vg2_1	25	0.02	94	1283
vg2_2	25	0.34	690	36620
vg2_3	25	0.05	86	1238
table5_2	17	0.95	2466	143655
table5_9	17	0.09	119	15817
table3_4	14	0.19	1351	19318
table3_1	14	0.16	2548	19606
duke2_2	22	0.12	218	40068
duke2_6	22	0.05	143	6882
misex3_1	14	0.16	362	16405
misex3c_13	14	0.12	824	18353
alu4_3	14	0.10	1232	4944

CONCLUSION

In this paper, the NP and NPN invariant properties of the FPRM weight vectors in terms of its number of products and literals have been analyzed with the aid of the ETDD data structure. An efficient method for computation of these weight vectors with reduced memory constraint has also been presented. One of the possible applications of the Reed–Muller weight and literal vectors classification with appropriate manipulation of these vectors similar to the method based on related Arithmetic transform [4], is that they can by used as filters for pruning the search space in the process of Boolean matching. The performance of a filter depends on both its capability of pruning and its computational cost [4,6,19]. We have presented several novel and effective Boolean matching filters composed of the selected row and column sums of Reed–Muller polarity coefficient matrix. The advantage of our first method is that aliasing NP or NPN equivalent classes can be resolved by expanding their signatures to include more FPRM weight components. The cell library can be preprocessed to store one unique characteristic string representing the signature for each NP or NPN representative function. The characteristic signatures represented by binary strings can be further compressed by run-length coding for large library. We believe that the

results given in this paper are useful for discovering new approaches to logic comparison problems encountered in logic synthesis process.

References

[1] Chang, C.H. and Falkowski, B.J. (1997) "Efficient symbolic computation of generalized spectra", *IEE Electron. Lett.* **33**(22), 1837–7838.

[2] Chang, C.H. and Falkowski, B.J. (1998) "Adaptive exact optimization of minimally testable FPRM expansions", *IEE Proc. Comput. Digit. Tech.* **145**(6), 385–394.

[3] Chang, C.H. and Falkowski, B.J. (1999) "Reed–Muller weight and literal vectors for NPN classification", *Proc. 32nd IEEE Int. Symp. Circuits Systems (ISCAS'99), Orlando, USA* **1**, 379–382.

[4] Chattopadhyay, S., Roy, S. and Pal Chaudhuri, P. (1994) "KGPMAP: library-based technology-mapping technique for antifuse based FPGA", *IEE Proc. Comput. Digit. Tech.* **141**(6), 361–368.

[5] Davio, M., Deschamps, J.P. and Thayse, A. (1978) *Discrete and Switching Functions* (McGraw-Hill, New York).

[6] DeMicheli, G. (1994) *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, New York).

[7] Edwards, C.R. (1975) "The application of Rademacher–Walsh transform to Boolean function classification and threshold-logic synthesis", *IEEE Trans. Comput.* **24**, 48–62.

[8] Fisher, L.T. (1974) "Unateness properties of AND-Exclusive-OR logic ciruits", *IEEE Trans. Comput.* **23**(2), 166–172.

[9] Green, D.H. (1986) *Modern Logic Design* (Addison-Wesley, Wokingham, MA).

[10] Harrison, M.A. (1965) *Introduction to Switching and Automata Theory* (McGraw-Hill, New York).

[11] Hurst, S.L. (1978) *The Logic Processing of Digital Signals* (Crane Russak, New York).

[12] Lai, Y.T., Pedram, M. and Vrudhula, S.B.K. (1994) "EVBDD-based algorithms for integer linear programming, spectral transformation and functional decomposition", *IEEE Trans. Comput.-Aid. Design Integ. Circ. Syst.* **13**(8), 959–975.

[13] Rahardja, S. and Falkowski, B.J. (1997) "Classification and graph-based representations of switching functions using a novel complex spectral technique", *Int. J. Electronics* **83**(6), 731–742.

[14] Staff of the Harvard Computation Laboratory (1951) *Synthesis of Electronic Computing Cirucits* (Harvard University Press, Cambridge, MA).

[15] Sasao, T., Fujita, M., eds, (1996) "Representation of Discrete Functions", (Kluwer Academic, Boston).

[16] Stankovic, R.S. and Falkowski, B.J. (2000) "Spectral interpretation of the fast tabular technique for fixed-polarity Reed–Muller expressions", *Int. J. Electronics* **87**(6), 641–648.

[17] Tekalp, A.M. (1995) *Digital Video Processing* (Prentice-Hall, Upper Saddle River).

[18] Tsai, C.C. and Marek-Sadowska, M. (1997) "Boolean functions classification via fixed polarity Reed–Muller forms", *IEEE Trans. Comput.* **46**(2), 173–186.

[19] Zhu, K. and Wong, D.F. (1993) "Fast Boolean matching for field-programmable gate arrays", *Proc. IEEE Euro. Design Automat. Conf.*, 352–357.

Authors' Biographies

Chip-Hong Chang received the Beng degree from National University of Singapore, and the M.Eng. and Ph.D. degrees in Electrical and Electronic Engineering from Nanyang Technological University, Singapore. He is currently an Assistant Professor at the School of Electrical and Electronic Engineering and Deputy Director of the Centre of High Performance Embedded Systems, Nanyang Technological University. His research interests include optimization and synthesis of VLSI digital circuits, graph theory, symbolic algorithms, and formal verification. He has published over 40 refereed journal and conference articles in these areas. He is a member of the IEEE.

Bogdan J. Falkowski received the MSEE degree from Technical University of Warsaw, Poland and the Ph.D. degree in Electrical and Computer Engineering from Portland State University, Oregon, USA. His industrial experience includes research and development positions at several companies. He then joined the Electrical and Computer Engineering Department at Portland State University. Since 1992 he has been with the School of Electrical and Electronic Engineering, Nanyang Technological University in Singapore where he is currently an Associate Professor. His research interests include VLSI systems and design, switching circuits, testing, and design of algorithms. He specializes in the design of digital circuits with the use of spectral methods and has published three book chapters and over 180 referred journal and conference articles in this area. He is a senior member of the IEEE, member of international advisory committee for International Conference on Applications of Computer Systems, guest editor of VLSI Design journal, and technical chair for IEEE International Conference on Information, Communication and Signal Processing hold in December 1999.