VLSI DESIGN
1999, Vol. 9, No. 2, pp. 135–146
Reprints available directly from the publisher
Photocopying permitted by license only

An Efficient Algorithm for the Calculation of Generalized Adding and Arithmetic Transforms from Disjoint Cubes of Boolean Functions

BOGDAN J. FALKOWSKI [a,*] and CHIP-HONG CHANG [b]

[a] *School of Electrical and Electronic Engineering, Nanyang Technological University, Block S1, Nanyang Avenue, Singapore 639798;*
[b] *Electronics Design Centre, French Singapore Institute, Nanyang Polytechnic, 180 Ang Mo Kio Ave 8, Singapore 569830*

(Received 21 April 1997; In final form 5 February 1998)

A new algorithm is given that converts a reduced representation of Boolean functions in the form of disjoint cubes to Generalized Adding and Arithmetic spectra. Since the known algorithms that generate Adding and Arithmetic spectra always start from the truth table of Boolean functions the method presented computes faster with a smaller computer memory. The method is extremely efficient for such Boolean functions that are described by only few disjoint cubes and it allows the calculation of only selected spectral coefficients, or all the coefficients can be calculated in parallel.

Keywords: Logic design, generalized Arithmetic and Adding transforms, Boolean functions, disjoint cubes, spectral techniques

I. INTRODUCTION

Manipulations and calculations of discrete functions is a fundamental task in many areas of Computer Science and Engineering ranging from applied mathematics, linear integer programming to artificial intelligence and in Computer-Aided Design of digital circuits [1–7]. For example, efficiently answering questions about satisfiability, tautology, equivalence and classification of Boolean functions has immense applications in various problems of CAD such as synthesis, verification, testing, library matching *etc.* It has been shown [1, 8, 9] that many of the above tasks can be efficiently performed when original Boolean domain is converted to standard arithmetic operations. Moreover, in applied investigations methods, Arithmetic and Adding transforms are formed step by step and the tools to solve tasks for separate stages have been developed. Application of Arithmetic transform to artificial intelligence can be illustrated by the examples of *arithmetic*

*Corresponding author. Tel.: (65)799-1327, Fax: (65)791-2687, e-mail: bogdanf@computer.org

predicates introduced by Hunt [10] that were also used for development of description for reliability of engineering systems [1, 7]. A search on trees and graphs is often used in expert systems. There are many ways to generate and store graph [11]. Description of a class of graphs by linear Arithmetic expansion can be useful in a number of cases [4].

The need for working without traditional Boolean operators (NOT, OR, AND, EXOR, NOR, NAND) can also be found in the area of a stochastics (probabilistics) of Boolean analysis [5, 7]. The main practical reason for a big interest in representing Boolean functions written as polynomials with standard symbols "+", "−", and "•" for addition, subtraction, and multiplication, respectively, is the easy way of the calculation of the expected value of such a polynomial. It is especially the case when the variables are stochastically independent of each other. Boolean polynomials are nothing else than the orthogonal expansions of the corresponding Boolean functions by the set of basis functions from an Arithmetic transform [5, 12, 13]. It should be noted, however, that the standard definition of such basis functions does not allow on the negation of any variables of Boolean functions and it is the reason why all the variables in Boolean polynomials are in affirmation [12]. When Boolean polynomials were used in testing, only variables in affirmation were considered [3, 14, 15]. Links between Reed–Muller and Arithmetic transforms for all not negated variables (so called "zero' polarity of both transforms) were investigated in [5, 13]. Since Reed–Muller transform of an n-variable Boolean function is known to have 2^n different polarities (such a transform is called a Generalized transform) [16, 17], the same concept was applied to Arithmetic transform and the Generalized Arithmetic transform was introduced in [18]. Another counterpart of Generalized Reed–Muller transform based only on addition of real numbers is a Generalized Adding transform introduced in [18]. The Adding transform has been found useful in a recently proposed signature-based library search mapper that is able to distinguish the nonequivalent functions with very low aliasing [19].

By using Generalized Arithmetic transform in an optimal polarity for a given Boolean function the number of terms in Boolean polynomial can be minimized which results in faster calculation of the value of such a polynomial. Generalized Adding transform is also useful when the incompletely specified Boolean functions are to be represented in the form of Generalized Reed–Muller expansions [16, 17]. When such a case occurs, it is impossible to retrieve the information about the don't care minterms from the Generalized Reed–Muller transform. On the other hand using additional information about incompletely specified Boolean function in the form of Generalized Adding transform allows to have all the information about the original Boolean function and the don't care minterms can be obtained back when they are needed. The latter case can occur in the decomposition of systems of Boolean functions.

As mentioned above Arithmetic and Adding transforms find applications in various problems of CAD and in probabilistics of Boolean analysis and testing. In many such applications the values of only some spectral coefficients are needed which is for example in the case of testing [3]. Therefore, there exists a need for efficient way of calculation for both transforms that has the following properties: ability to evaluate only some chosen spectral coefficients, operations on both completely and incompletely specified Boolean functions, and the usage of efficient reduced representation of Boolean functions to calculate spectrum.

Recently, the methods have been shown which operate on various decision diagrams to calculate and store the resulting Arithmetic and Adding spectra [20, 21]. These methods can be used efficiently in various CAD systems and the decision diagrams can represent the original Boolean functions and their Arithmetic spectra. Moreover, with detailed investigations of operations on Boolean functions and variables in spectral domain of Arithmetic transform [22],

such operations may be performed directly on decision diagrams.

Different decision diagrams [6, 20, 21, 23] have proved to be very convenient data structures for majority of discrete functions representations permitting manipulations and calculation with large discrete functions efficiently in terms of space and time. Therefore they are frequently used to represent data structures in modern CAD VLSI systems. However, some of such systems are based on cubical representation [23, 24] rather than decision diagrams and the current article solves the problem of efficient calculation of generalized Arithmetic and Adding spectra for such CAD systems.

The presented algorithm has all the properties required in CAD applications of Arithmetic and Adding spectra and calculates both spectra in an efficient way. The first attempt to overcome the inefficiency of the calculation of both spectra directly from the definition of the transforms by matrix multiplication was done in [25]. The algorithm presented there solved the problem of calculating the spectra of incompletely specified Boolean functions. It allowed also on the independent calculation of only some chosen spectral coefficients. However the algorithm presented there still operated on truth tables. The new algorithm presented in this article has addressed and solved this issue. By allowing to represent the Boolean function in the form of an array of disjoint cubes instead of minterms, the spectral coefficients can be computed more rapidly from such a reduced representation with smaller required memory, while the ability to calculate only partial spectra is still preserved. Hence, the new algorithm has allowed practical applications of both Adding and Arithmetic transforms in real life problems' sizes of analysis, synthesis, testing and probabilistics of Boolean functions for CAD systems using cubical rather than graph based representations of discrete functions.

The algorithm introduced in the following section is not only valid for completely specified Boolean functions but also for incompletely specified functions, since don't care minterms can also be represented in the form of disjoint cubes as input data to the algorithm. In order to use Boolean functions that are represented as minterms or arrays of non disjoint cubes, the input data are preprocessed by a fast algorithm that generates on array of disjoint ON-cubes (in the case of completely specified Boolean functions) or disjoint ON- and DC-cubes (in the case of incompletely specified functions). The algorithm that generates such an array and its implementation is described in [26]. For each disjoint cube, the appropriate partial spectral coefficients from either Adding or Arithmetic spectrum is calculated. Adding spectrum is found by adding all the entries corresponding to the complete array of disjoint cubes defining a given Boolean function. For Arithmetic spectrum, the final addition/subtraction is performed based on some rules that describe the relationship between the disjoint cubes and partial arithmetic spectral coefficients.

II. BASIC DEFINITIONS AND PROPERTIES

The properties of disjoint cube representation of Boolean functions used in the description of the algorithm calculating Adding and Arithmetic spectra are stated. In what follows, the definitions and basic properties of Adding and Arithmetic transforms are presented to make the paper self contained.

DEFINITION 1 The *literal* $\dot{x}$ of a variable x of a Boolean function can be either in affirmative (x) or negative form ($\bar{x}$).

DEFINITION 2 The *polarity number* of a spectrum, denoted by ω, is a binary string computed by taking the n bit straight binary code (SBC) formed by writing a 0 or a 1 for each literal according to whether this literal is used in affirmative or negative form in all the terms.

Example 1 The polarity number of the term $x_1x_2\bar{x}_3x_4 = 0010$.

DEFINITION 3 A cube is represented by a *positional notation* where "0" corresponds to a negated literal, "1" to the affirmative literal and "X" to the literal that is absent in a cube.

Example 2 The cube $x_1\bar{x}_3x_4$ is represented by $1X01$.

DEFINITION 4 A *cube of degree m* is a cube that has m defined literals that can be either affirmative or negative (*i.e.*, m is equal to the sum of the number of zeros and ones in the description of a cube).

Let symbol p denotes the number of X's in the cube and n denotes the number of variables of a given Boolean function F. Then, $n = m+p$.

DEFINITION 5 The matrix of order $N = 2^n$ for *Adding Transform* in polarity ω is defined as [18, 20–22, 25]:

$$AD_N^{<\omega>} = \bigotimes_{r=1}^{n} AD_2^{<\omega_r>} \tag{1}$$

where the elementary Adding matrices of order 2 for polarity bits 0 and 1 are $AD_2^{<0>} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $AD_2^{<1>} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ respectively; "$\otimes$" is the right hand Kronecker product and ω_r is the r-th bit in the binary representation of the polarity number ω.

DEFINITION 6 The matrix of order $N = 2^n$ for *Arithmetic Transform* in polarity ω is defined as [18, 20–22, 25]:

$$AR_N^{<\omega>} = \bigotimes_{r=1}^{n} AR_2^{<\omega_r>} \tag{2}$$

where the elementary Arithmetic matrices of order 2 for polarity bits 0 and 1 are $AR_2^{<0>} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$ and $AR_2^{<1>} = \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix}$; "$\otimes$" is the right hand Kronecker product and ω_r is the r-th bit in the binary representation of the polarity number ω.

Property 1 Both matrices $AD_N^{<\omega>}$ and $AR_N^{<\omega>}$ are inverses of each other for all polarities ω.

The set of all 2^n Arithmetic and Adding transformation matrices and corresponding spectra for all possible polarities ω for an n-variable Boolean function are called *Generalized* or *Multipolarity Arithmetic and Adding Transforms* and *Spectra* accordingly [18].

Two types of coding are used for the truth vector of a Boolean function before its spectrum is computed [27]. In the S coding, *false* minterms are represented by 1, *true* minterms by -1 and *don't care* (DC) minterms by 0. In the R coding the truth vector is represented by its original values: 0 for *false* minterms and 1 for *true* minterms. The DC minterms are represented by 0.5.

The adding spectrum for both codings can be calculated by the following matrix multiplication.

$$ADS_N = [AD_N] \times FS_N \tag{3}$$

$$ADR_N = [AD_N] \times FR_N \tag{4}$$

In the above equations, the following symbols have been used: AD_N denotes Adding transform matrix of order N; FS_N and FR_N denote Boolean function vectors of size $N=2^n$ in the S coding and R coding respectively where n is the number of variables; ADS_N and ADR_N denote vectors representing Adding spectrum for the S coding and R coding respectively. The complete spectrum is denoted by capital letters. A spectral coefficient is denoted by a small letter with a subscript called an *index*. For example, the symbol ads_I denotes an adding spectral coefficient from the spectrum calculated for S coding that has an index I. The symbol of a spectral coefficient for both transforms is a_I where the letters 'd' and 'r' are absent in the symbol.

The analogous equations are valid for Arithmetic transform by replacing all symbols 'AD' by 'AR' in (3) and (4). The meaning of all the symbols described above is still applicable with the understanding that the word 'Adding' is replaced by 'Arithmetic'.

DEFINITION 7 Each spectral coefficient from a spectrum is described by its *order*, and *subindices*. The order of a spectral coefficient is equal to the number of subindices in the index, where the subindices are the subscripts of all variables of a standard characteristic set corresponding to the coefficient.

Property 2 The number of spectral coefficients of order z is equal to $C_z^n = \binom{n}{z}$ where n is the number of variables of a Boolean function.

The definition of a standard characteristic set, independent of the type of the transform, is given below.

DEFINITION 8 There exists a correlation value between a Boolean function F and the *standard characteristic set* u corresponding to each spectral coefficient a_I. The standard characteristic set is given by the set of subindices which are present in the index I of the spectral coefficient a_I.

The index I of the spectral coefficient can also be represented by a positional notation where "1" corresponds to the literal that is present in the index and "0" to the literal that is absent from the index.

The standard characteristic set obtained by direct matrix multiplication using Eqs. (3) and (4) is in the order that by similarity to known orders of Walsh functions can be called a Hadamard order [12, 27]. By the properties of orthogonal spectral transforms [3, 12, 16–18], the superset of standard characteristic sets is canonic.

Example 3 The standard characteristic set u of the second order spectral coefficient $a_{x_1x_2}$ for a Boolean function $F(x_1, x_2, x_3)$ is $u = \{x_1, x_2\}$. The index of this spectral coefficient in the positional notation is $I = 110$.

Property 3 For a given *polarity* ω of the spectrum, all standard characteristic sets have the *same* polarity number and each variable in the standard characteristic set can be either in affirmative form, negative form or be absent from the set.

For an n variable Boolean function, there are 2^n different polarities and consequently there are 2^n different canonical Arithmetic and Adding expansions of a Boolean functions.

III. THE GENERALIZED SET COMPARISON METHOD

With the definitions given in Section 2, the Generalized Set Comparison Method to calculate Adding and Arithmetic spectra can be described. Since there exists a linear relationships between spectra in both S and R codings [18, 28], it is not necessary to consider both codings in this article. In the continuation, all the derivations are presented only for the spectrum in R coding. To obtain the spectrum in S coding from the spectrum in R coding using the linear relationship or by directly modifying the operation presented later in this article is trivial. In order to compute both Adding and Arithmetic spectra, two new operations: cube adding and arithmetic operators are introduced. The presented properties and definitions are based on behavior of Boolean functions and operations in Arithmetic spectral domain proven in [22]. The correctness of the algorithm has also been verified by extensive experimentation and generation of Arithmetic and Adding spectra for all Boolean functions for $n < 5$.

DEFINITION 9 A *cube adding operator*, denoted as χ_{ad}, is defined as an operation between bits of an index of an adding spectral coefficient and a cube in positional notation by Table I(a). The result of this operator is a three valued string (a ternary string).

TABLE I(a) Cube adding operator between bits of a cube and an index in positional notation

χ_{ad}	Index I	
Cube	0	1
0	1	1
1	0	1
X	1	2

DEFINITION 10 A *cube arithmetic operator*, denoted as χ_{ar} is defined as an operation between bits of an index of an arithmetic spectral coefficient and a cube in positional notation by Table I(b). χ_{ar} operator is analogous to χ_{ad} operator except that for the "X" bits of the cube the result of χ_{ar} operation is equal to the complemented value of the corresponding bit of the index. Hence, such a result is a two valued string (a binary string).

DEFINITION 11 The *partial spectral coefficients* of an ON- or DC- cube cu_j of degree m from disjoint cube representation of a Boolean function F are those part of the final spectral coefficients a_I that correspond to the contribution of this cube cu_j to the full n-space spectrum of the Boolean function F described by the array of all the disjoint cubes.

Property 4 The partial adding spectral coefficient, adr_I contributed by an ON-cube, cu_j is obtained by the multiplication of all the elements of the ternary string resulting from the operation $cu_j \chi_{ad} I$.

Property 5 The partial spectral coefficient arr_I contributed by an ON-cube cu_j is obtained by the multiplication of all the elements of the binary string resulting from the operation $cu_j \chi_{ar} I$. The value of the partial spectral coefficient has to be negated if the number of affirmative literals describing the cube is even. The final value of such a coefficient has to be negated again if the order of the spectral coefficient is even.

Properties 4 and 5 are valid only for zero polarity ω for which each literal in the polarity is affirmative. Properties 4 and 5 are valid for any other polarity ω provided the cube has been adjusted to ω prior to the respective cube adding or cube arithmetic operation. The explanation of the cube adjustment operation is presented later in Definition 12.

Property 6 The partial spectral coefficient contributed by a DC-cube is equal to one half of the contribution of an ON-cube that has the same positional notation. Additionally, the *number of partial spectral coefficients* (*npsc*) describing the Boolean function F is equal to the number of ON- and DC-cubes describing this function.

Property 6 is valid for both adding and arithmetic spectra. It is also obvious that the number *npsc* does not have a canonical value for a given Boolean function and depends on the disjoint cube representation of such a function obtained by Algorithm from [26].

Example 4 The *ADR* spectrum in zero polarity ($\omega = 000$) for the three-variable completely specified function $F(x_1, x_2, x_3) = x_2\bar{x}_3 + \bar{x}_1 x_2 x_3$ is calculated in Table II.

The first row of Table II gives all possible spectral coefficients adr_I according to their order. To abbreviate the notation used, the members of the standard characteristic set of a given spectral coefficient are listed as the subindices of such a coefficient. For example, a spectral coefficient $adr_{x_1x_3}$ is denoted by adr_{13}. The binary representation of the indices of the spectral coefficients is given in the second row. In the first column the two disjoint cubes of F are shown in their binary representation. The values of the partial spectral coefficients for each cube are computed using the cube adding operation and Property 4. The final values of the spectral coefficients are obtained by adding the respective partial values in the columns.

TABLE I(b) Cube arithmetic operation between bits of a cube and an index in positional notation

χ_{ad}	Index I	
Cube	0	1
0	1	1
1	0	1
X	1	0

TABLE II *ADR* spectrum of $F(x_1, x_2, x_3) = x_2\bar{x}_3 + \bar{x}_1 x_2 x_3$ for $\omega = 000$

Cube	adr_0	adr_1	adr_2	adr_3	adr_{12}	adr_{13}	adr_{23}	adr_{123}
	000	100	010	001	110	101	011	111
X10	0	0	1	0	2	0	1	2
011	0	0	0	0	0	0	1	1
Result	0	0	1	0	2	0	2	3

Example 5 As an illustration, the partial spectral coefficient $adr_{x_1x_2}$ is computed for the cube $x_2\bar{x}_3$.

$$X10\chi_{ad}110 = 211.$$

The value of the partial spectral coefficient $adr_{x_1x_2} = 2 \times 1 \times 1 \times 1 = 2$.

While the value of the adding spectral coefficient is in the range from 0 to 2^n, an arithmetic spectral coefficient can have a negative value. This is apparent from Formula (2) as the positions of the -1 entries in Arithmetic transformation matrix determine which spectral coefficients can be negative.

Example 6 The *ARR* spectrum in zero polarity for the function F given in Example 4 is calculated in Table III.

The description of Table III is identical to that of Table II. The only difference is the addition of a new column (labeled as *Coding*) next to the disjoint cube and a new row (labeled as *Final ARR*) below the value to indicate the necessary negation according to Property 5.

The previous Properties 4 and 5 are valid only for the positive polarity. They can be, however, adopted for all polarities with a cube adjustment operation defined below.

DEFINITION 12 The *cube adjustment operator* ($CU_{\oplus}$) on two cubes is essentially a bitwise modulo-2 addition except that "X" is invariant under such operation. Table IV shows the cube adjustment operator between bits of a cube and a polarity number.

Example 7 The cube $cu_1 = 1X00$ adjusted to a polarity number $\omega = 1010$ is given by $1X00$ $CU_{\oplus}1010 = 0X10$.

When all ON- and DC-cubes for a given function have been adjusted to the required polarity number, Properties 4 to 6 are used to calculate the partial and total Adding and Arithmetic spectra.

IV. IMPLEMENTATION OF THE ALGORITHM FOR THE CALCULATION OF *ADR* AND *ARR*

The algorithm presented below refers to properties and definitions from the previous section. In order to use Boolean functions that are represented as minterms or arrays of non disjoint cubes, the input data to the new algorithm are preprocessed by the fast algorithm [26] that generates an array of disjoint ON- and DC-cubes. The main body of the algorithm is a *for loop* where the values of the partial coefficients corresponding to an adjusted cube are calculated. The total number of iterations is equal to *npsc*. During execution, the values of the spectral coefficients of a currently processed cube are added to the already calculated values corresponding to the previous cubes. Therefore, this algorithm requires only one spectral coefficient to be kept in the computer memory at a time. When the coefficients are calculated separately or in a group, there is no limit on the number of variables n since the calculated coefficient or group of coefficients can be transferred immediately to the hard disk. This is, however, performed at the expense of the processing time.

Owing to the property that the values for spectral coefficients of even orders have to be negated for the *ARR* transform, the spectrum is calculated order by order when full spectrum is required. To avoid the calculation of the number

TABLE III *ARR* spectrum of $F(x_1, x_2, x_3) = x_2\bar{x}_3 + \bar{x}_1x_2x_3$ for $\omega = 000$

Cube	Coding	arr_0	arr_1	arr_2	arr_3	arr_{12}	arr_{13}	arr_{23}	arr_{123}
		000	100	010	001	110	101	011	111
X10	1	0	0	1	0	0	0	1	0
011	-1	0	0	0	0	0	0	-1	-1
Result		0	0	1	0	0	0	0	-1
Final *ARR*		0	0	1	0	0	0	0	-1

TABLE IV Cube adjustment operator between bits of a cube and a polarity number in positional notation

$CU_{\oplus}$	Polarity number ω	
Cube	0	1
0	0	1
1	1	0
X	X	X

of affirmative literals in the index of each spectral coefficient, the coefficients are generated in ordering corresponding to Rademacher ordering of Walsh functions [27]. It is, however, always possible to calculate any coefficient from any order independently. The following notations are used for the algorithm.

ω	polarity number of the spectrum
cube	disjoint ON- or DC-cube of the input function
I	index of the spectral coefficient currently processed
sign	sign of the *ARR* spectrum determined according to Property 5
temp	temporary storage for the partial spectral coefficient

Algorithm 1 Generation of the *ADR* and *ARR* Spectral Coefficients from Disjoint Cubes

```
Begin
    adjust each cube by
        cube := cube CU⊕ω;
    do for each spectral coefficient {
        value := 0;
        do for each cube {
            sign := 1;
            if number of ones in the cube is even and ARR spectrum is selected
                sign := −1;
            if ARR spectrum is selected
                temp := mult(cube χar I);
            else
                temp := mult(cube χad I);
            if ON-cube
                value := value + temp × sign;
            else
                value := value + temp × sign × 0.5;
        }
        if ARR spectrum is selected and order of spectral coefficient is even
            value := value × −1;
        return value of the spectral coefficient;
    }
End

Subroutine mult
  Begin
      retval := 1;
      do for each bit of argument {
          case bit {
```

```
            0: retval := 0;
                return retval;
            1: break;
            2: retval := shift left retval by 1 bit;
        }
    }
    return := retval;
End
```

value	value of the spectral coefficient currently calculated for all cubes
mult	subroutine to multiply the elements of either binary or ternary string
argument	binary or ternary string resulting from cube arithmetic or adding operation
retval	return value of the subroutine *mult*

Example 8 An array of disjoint ON- and DC-cubes for an incompletely specified Boolean function of four variables is given in Table V. Polarity number 1101 is selected for the computation of both spectra. In the third column of Table V, the original cubes in the first column are replaced by the results of the cube adjustment operation. The adjusted cubes are taken to generate both Adding and Arithmetic spectra. The coding for Adding spectrum is identical to the original R coding and is given as the fourth column in Table V. The new coding for Arithmetic spectrum is determined by Properties 5 and 6 and is given as the fifth column of Table V.

The adjusted cubes and its coding are transferred to Table VI where each final spectral coefficient is calculated for the whole function in turn. Each row of Table VI, that starts with a cube, describes the values of the partial spectral coefficients of such a cube. The final value of Adding spectrum for the whole function is obtained after adding up all the partial spectra and the result is shown as the bottom row of Table VI.

In Table VII, the calculation of the *ARR* spectrum for the same function and same polarity is shown. The only differences are that the signs of the even order spectral coefficients have to be negated according to Property 5 after adding up all the partial spectra. The final result of the calculation of Arithmetic spectrum is given in the bottom row of Table VII.

TABLE V Disjoint cubes of the incompletely specified function in Example 8 adjusted to $\omega = 1101$

Cube	Type	Result of cube adjustment operation	Coding for *ADD* spectrum	Coding for *ARR* spectrum
1X00	ON	0X01	1	1
1X1X	ON	0X1X	1	1
0001	ON	1100	1	−1
0X11	DC	1X10	0.5	−0.5
0000	DC	1101	0.5	0.5

V. EXPERIMENTAL RESULTS AND COMPARISONS

The new algorithm was implemented in C++ and run on a Sun SPARC IPC 2/475 station. Execution time for the calculation of the whole spectrum for polarities 00...0 and 00...010110 for the functions from the MCNC industrial benchmarks are given in Table VIII. The functions have been preprocessed by algorithm from [26] to obtain their disjoint representation before calculating their spectra. The number of disjoint cubes obtained and the time required for their calculation are given in the fourth and fifth columns, respectively. In the second and third columns, the number of input/output variables of the benchmark functions are given. The type of spectrum is abbreviated by *adr* and *arr* for adding and

TABLE VI *ADR* spectrum of the incompletely specified function in Example 8 for $\omega = 1101$

Adjusted Cube	Coding	0000 adr_0	1000 adr_1	0100 adr_2	0010 adr_3	0001 adr_4	1100 adr_{12}	1010 adr_{13}	1001 adr_{14}	0110 adr_{23}	0101 adr_{24}	0011 adr_{34}	1110 adr_{123}	1101 adr_{124}	1011 adr_{134}	0111 adr_{234}	1111 adr_{1234}
0X01	1	0	0	0	0	1	0	0	1	0	2	1	0	2	1	2	2
0X1X	1	0	0	0	1	0	0	1	0	2	0	2	2	0	2	4	4
1100	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	1
1X10	0.5	0	0	0	0	0	0	0.5	0	0	0	0	1	0	0.5	0	1
1101	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0.5
Result		0	0	0	1	1	1	1.5	1	2	2	3	4	3.5	3.5	6	8.5

TABLE VII *ARR* spectrum of the incompletely specified function in Example 8 for $\omega = 1101$

Adjusted Cube	Coding	0000 arr_0	1000 arr_1	0100 arr_2	0010 arr_3	0001 arr_4	1100 arr_{12}	1010 arr_{13}	1001 arr_{14}	0110 arr_{23}	0101 arr_{24}	0011 arr_{34}	1110 arr_{123}	1101 arr_{124}	1011 arr_{134}	0111 arr_{234}	1111 arr_{1234}
0X01	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0
0X1X	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
1100	−1	0	0	0	0	0	−1	0	0	0	0	0	−1	−1	0	0	−1
1X10	−0.5	0	0	0	0	0	0	−0.5	0	0	0	0	0	0	−0.5	0	0
1101	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0.5
Result		0	0	0	1	1	−1	0.5	1	0	0	1	−1	−0.5	0.5	0	−0.5
Final *ARR*		0	0	0	1	1	1	−0.5	−1	0	0	−1	−1	−0.5	0.5	0	0.5

TABLE VIII Execution time for the MCNC benchmark functions

Function	Number of inputs	Number of outputs	Number of disjoint cubes	Time for disjoint cubes calculation	Execution time for spectra calculation			
					$\omega = 00...000$		$\omega = 00...010110$	
					adr	*arr*	*adr*	*arr*
rd73	7	3	127	0.3*u* 0.0*s*	0.4*u* 0.0*s*	0.4*u* 0.0*s*	0.4*u* 0.1*s*	0.4*u* 0.0*s*
inc	7	9	33	0.0*u* 0.0*s*	0.5*u* 0.0*s*	0.5*u* 0.0*s*	0.5*u* 0.1*s*	0.5*u* 0.0*s*
5xpl	7	10	68	0.1*u* 0.0*s*	0.5*u* 0.0*s*	0.4*u* 0.0*s*	0.5*u* 0.1*s*	0.4*u* 0.0*s*
misex1	8	7	17	0.0*u* 0.0*s*	0.4*u* 0.0*s*	0.3*u* 0.1*s*	0.4*u* 0.0*s*	0.2*u* 0.1*s*
9sym	9	1	145	0.4*u* 0.0*s*	1.4*u* 0.1*s*	1.4*u* 0.1*s*	1.5*u* 0.1*s*	1.4*u* 0.1*s*
clip	9	5	151	0.5*u* 0.0*s*	2.3*u* 0.2*s*	2.1*u* 0.1*s*	2.5*u* 0.1*s*	2.2*u* 0.1*s*
sao2	10	4	93	0.2*u* 0.1*s*	3.2*u* 0.2*s*	2.7*u* 0.1*s*	3.1*u* 0.2*s*	2.6*u* 0.2*s*
misex3	14	14	1295	67.3*u* 0.2*s*	1063.3*u* 5.0*s*	951.4*u* 4.6*s*	1151.4*u* 4.8*s*	990.9*u* 4.4*s*
b12	15	9	70	0.7*u* 0.1*s*	160.7*u* 6.4*s*	99.5*u* 6.1*s*	165.5*u* 6.4*s*	99.7*u* 5.8*s*

arithmetic spectrum, respectively. The execution time is given in seconds where the symbol *u* denotes the elapsed user time and the symbol *s* denotes the elapsed system time.

As the number of spectral coefficients increase exponentially with the number of input variables, the execution time is affected inevitably by the number of input and output variables, but mainly depends on the number of disjoint cubes. However, as the algorithm uses disjoint cubes instead of minterms, the storage requirement also increases linearly with the number of disjoint cubes, as shown in Table VIII. The computational complexity of the above algorithm is of the order $O(m)$ where m is the number of disjoint cubes. Due to the property that the cube arithmetic operation generates more zero partial spectral coefficients than the cube adding operation, the execution time for the arithmetic spectrum is generally faster than that for the adding spectrum.

In the literature, two other spectra of Boolean functions have been calculated from disjoint cubes: Walsh spectrum [27] and Reed–Muller spectrum [16, 29, 30]. It is obvious that the efficiency of all these methods depends on the function in question and by using different mathematics even for the

same original functions, the number of spectral coefficients not equal to zero differs. Hence it is difficult to compare meaningfully the storage requirements, but it can be observed from the published results that the execution time is similar for the functions with comparable number of disjoint cubes. Moreover, each of the methods based on disjoint cubes can calculate spectral coefficients in groups or even individually. When the coefficients are calculated separately, the number of input and output variables is virtually unlimited, since the coefficients can be stored in groups on the hard disk. This is, however, traded off for the increased processing time.

VI. CONCLUSION

A new algorithm that generates any Generalized Adding and Arithmetic spectra from the disjoint cube representation of Boolean functions has been shown. Up to now, the computation of both spectra by known algorithms requires the representation of the Boolean function in the form of a truth table composed of minterins [18, 25] or by using different decision diagrams [20, 21]. By contrast, in this paper the spectrum is generated directly from the reduced representation of Boolean functions in the form of arrays of disjoint cubes [26]. Since the number of such cubes can be considerably smaller than the number of minterms, the memory requirements can be reduced significantly. The advantages of this kind of representation used frequently in modern CAD VLSI systems [23], especially the fact that for practical functions the number of disjoint cubes is much smaller than the number of minterms, has been manifested in [31]. It is also evident from Tables VI and VII that the number of disjoint cubes for the presented function is much smaller than the number of its minterms. The ability to calculate only some spectral coefficients made possible by this research is very important since there are many spectral methods in digital logic design for which the values of only selected spectral coefficients are needed [3, 27, 28].

The fundamental advantage of the presented algorithm is the usage of a reduced representation of Boolean functions in the form of disjoint cubes as the internal data from which the algorithm calculates the spectra. Such an approach gives the presented algorithm the ability to yield solutions to problems of very high dimensions and is applicable to these CAD systems which use cubical representation for discrete functions. The algorithm is very well suited for systolic VLSI realizations, and may be implemented as hardware coprocessor in a manner similar to those used for other binary expansions [4].

References

[1] Bennets, R. (1982). "Analysis of Reliability Block Diagrams by Boolean Techniques," *IEEE Trans. on Reliability*, **31**, 156–166.

[2] Falkowski, B. J., Shmerko, V. P. and Yanushkevich, S. N., "Arithmetical Logic–Its Status and Achievements," Invited Paper, *Proc. of the 4th Int. Conf. on Applications of Computer Systems*, Szczecin, Poland, pp. 208–223, Nov. 1997.

[3] Heidtmann, K. D. (1991). "Arithmetic Spectrum Applied to Fault Detection for Combinational Networks," *IEEE Trans. on Computers*, **40**(3), 320–324.

[4] Kukharev, G. A., Shmerko, V. P. and Yanushkevich, S. N. (1991). *Technique of Binary Data Parallel Processing for VLSI*, Minsk: University Press (in Russian).

[5] Kumar, S. K. and Breuer, M. A. (1981). "Probabilistic Aspects of Boolean Switching Functions via a New Transform," *Journal ACM*, **28**, 502–520.

[6] Sasao, T. and Fujita, M. (1996). *Representations of Discrete Functions*, Boston: Kluwer Academic.

[7] Schneeweiss, W. G. (1989). *Boolean Functions with Engineering Applications and Computer Programs*, Berlin: Springer-Verlag.

[8] Hammer, P. L. and Rudeanu, S. (1968). *Boolean Methods in Operational Research and Related Areas*, Berlin: Springer-Verlag.

[9] Ivanescu, P. and Rudeanu, S. (1966). *Pseudo-Boolean Methods for Bivalent Programming*, Lecture Notes in Mathematics, Vol. 23, Berlin: Springer-Verlag.

[10] Hunt, S. (1975). *Artificial Intelligence*, Reading: Addison-Wesley.

[11] Papadimitriou, C. H. (1994). *Computational Complexity*, Reading: Addison-Wesley.

[12] Falkowski, B. J. and Chang, C. H., "Generation of Multi-Polarity Arithmetic Transform from Reduced Representation of Boolean Functions," *Proc. of 28th IEEE Intnl. Symp. on Circuits and Systems*, Seattle, Washington, pp. 2168–2171, May 1995.

[13] Falkowski, B. J. and Chang, C. H., "Fast Generalized Arithmetic and Adding Transforms," *Proc. of 8th IFIP WG 10.5 Int. Conf. on Very Large Scale Integration*, Makuhari, Chiba, Japan, pp. 723–728, Sep. 1995.
[14] Parker, K. P. and McCluskey, E. J. (1975). "Analysis of Logic Circuits with Faults using Input Signal Probabilities," *IEEE Trans. on Computers*, **23**(5), 573–578.
[15] Parker, K. P. and McCluskey, E. J. (1975). "Probabilistic Transform of General Combinational Networks," *IEEE Trans. on Computers*, **24**(6), 668–670.
[16] Falkowski, B. J. and Perkowski, M. A., "On the Calculation of Generalized Reed–Muller Canonical Expressions from Disjoint Cube Representation of Boolean Functions," *Proc. of 33rd IEEE Midwest Symp. on Circuits and Systems*, Calgary, Alberta, pp. 1131–1134, August 1990.
[17] Falkowski, B. J. and Chang, C. H., "Generation of Fixed Polarity Reed–Muller Expansions from Subset of Walsh Spectral Coefficients for Completely Specified Boolean Functions," *Proc. of 5th Intnl. Workshop on Spectral Techniques*, Beijing, China, pp. 214–219, March 1994.
[18] Falkowski, B. J. and Perkowski, M. A., "A Family of All Essential Radix-2 Addition/Subtraction Multi-Polarity Transforms: Algorithms and Interpretations in Boolean Domain," *Proc. of 23rd IEEE Intnl. Symp. on Circuits and Systems*, New Orleans, Louisiana, pp. 2913–2916, May 1990.
[19] Chattopadhyay, S., Roy, S. and Pal Chaudhuri, P. (1994). "KGPMAP: Library-Based Technology Mapping Technique for Antifuse based FPGAs," *IEE Proc., Comput. Digit. Techn.*, **141**(6), 361–368.
[20] Falkowski, B. J. and Chang, C. H., "Efficient Algorithms for the Calculation of Arithmetic Spectrum from OBDD and Synthesis of OBDD from Arithmetic Spectrum for Incompletely Specified Boolean Functions," *Proc. of 27th IEEE Int. Symp. on Circuits and Systems*, London, United Kingdom, **1**, 197–200, May 1994.
[21] Falkowski, B. J. and Chang, C. H. (1997). "Calculation of Arithmetic Spectra from Free Binary Decision Diagrams," *Proc. of 30th IEEE Int. Symp. on Circuits and Systems*, Hong Kong, **3**, 1764–1767.
[22] Chang, C. H. and Falkowski, B. J. (1996). "Operations on Boolean Functions and Variables in Spectral Domain of Arithmetic Transform," *Proc. of 29th IEEE Int. Symp. on Circuits and Systems*, Atlanta, Georgia, **4**, 400–403.
[23] De Micheli, G. (1994). *Synthesis and Optimization of Digital Circuits*, New York: McGraw Hill.
[24] Hachtel, G. D. and Somenzi, F. (1996). *Logic Synthesis and Verification Algorithms*, Boston: Kluwer Academic.
[25] Falkowski, B. J., Schaefer, I. and Perkowski, M. A. (1992). "Generation of Adding and Arithmetic Multi-Polarity Transforms for Incompletely Specified Boolean Functions," *Int. Journal of Electronics*, **73**(2), 321–331.
[26] Falkowski, B. J., Schaefer, I. and Chang, C. H., "An Efficient Computer Algorithm for the Calculation of Disjoint Cubes Representation of Boolean Functions," *Proc. of 36th IEEE Midwest Symp. on Circuits and Systems*, Detroit, Michigan, pp. 1308–1311, August 1993.
[27] Falkowski, B. J., Schaefer, I. and Perkowski, M. A., "Effective Computer Methods for the Calculation of Rademacher-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions," *IEEE Trans. on Computer-Aided Design*, **11**(10), 1207–1226, October 1992.
[28] Karpovsky, M. G. (1976). *Finite Orthogonal Series in Design of Digital Devices*, New York: Wiley.
[29] Riege, M. W. and Besslich, Ph. W., "Low-complexity Synthesis of Incompletely Specified Multiple-output Mod-2 Sums," *IEE Proc.*, Pt. E, **139**(4), 355–361, July 1992.
[30] Varma, D. and Trachtenberg, E. A., "Computation of Reed–Muller Expansions of Incompletely Specified Boo-lean Functions from Reduced Representations," *IEE Proc.*, Pt. E, **138**(2), 85–92, March 1991.
[31] Ciesielski, M. J., Yang, S. and Perkowski, M. A., "Minimization of Multiple-Valued Logic based on Graph Coloring," Tech. Rep. TR-CSE-90-13, Dept. Elect. Comp. Eng., Univ of Massachusetts, Amherst, September 1990. (Earlier version of this paper appeared as: "Multiplevalued Minimization based on Graph Coloring," *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 262–265, October 1989).

Authors' Biographies

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. His research interests include VLSI systems and design, switching circuits, testing, and design of algorithms. He is a senior member of the IEEE, member of international advisory committee for International Conference on Applications of Computer Systems and technical chair for IEEE International Conference on Information, Communication and Signal Processing to be held in December 1999 in Singapore.

Chip-Hong Chang received the BEng degree from National University of Singapore, and the MEng and Ph.D. degrees in Electrical and Electronic Engineering from Nanyang Technological University, Singapore. He is currently a lecturer at Electronic Design Centre, French-Singapore Institute of Nanyang Polytechnic. His research interests include optimization and synthesis of VLSI digital circuits, graph theory and symbolic algorithms, and cryptographic systems. He is a member of the IEEE.