Computing and Informatics, Vol. 29, 2010, 1073–1088

CONDITIONS OF THE AFFINE EXTENSION OF AN INCOMPLETELY DEFINED BOOLEAN FUNCTION

Piotr Porwik

Institute of Computer Science, University of Silesia
Będzińska 39, 41-200 Sosnowiec, Poland
e-mail: piotr.porwik@us.edu.pl

Manuscript received 7 July 2008; revised 11 June 2009
Communicated by Elena Gramatová

Abstract. The paper presents conditions of extension of the weakly defined Boolean functions to their full affine form. The main goal of the analysis is a fast estimation whether a given incompletely defined function can be extended to a full affine form. If it is possible a simple algorithm of the states completion has been proposed. In such a case undefined points are clearly replaced by 0, 1 values. Spectral coefficients of a Boolean function allow to determine whether a partially defined function can be realised as affine.

Keywords: Walsh coefficients, coefficients distribution, incompletely defined Boolean function, affine function

Mathematics Subject Classification 2000: 06E30, 94C10, 43A32

1 INTRODUCTION

Incompletely defined Boolean functions are widely used in logic design because such forms are flexible and convenient in many practical applications. From incompletely, especially from weakly defined functions, different Boolean forms can be constructed, like Reed-Muller, affine, threshold, bent and so on.

In practical applications, linear operations can be realized with the aid of the EXclusive OR (EXOR) gates. Nowadays we can observe that the role of the EXOR gates in the design process is very important and most desirable. It is because such an approach offers an interesting compromise between testability, number of terms, area and speed [13, 17, 23]. Additionally, AND/EXOR logic often leads to a more compact realization of switching functions than AND/OR. It is especially suitable for arithmetic circuits: adders and parity checkers, multipliers and telecommunication circuits. A Boolean function can be represented by its so-called minterms expression – the Sum of Products (SOP) [8]. It has been shown that the Exclusive Sum of Products (ESOP) forms generally require fewer logic gates than traditional SOP. Hence it can be seen that the new EXOR gates technologies are more practical: the circuits can easily be mapped to modern Field Programmable Gate Arrays (FPGAs) [21, 8].

On the one hand, the linearity (nonlinearity) of functions is applied in cryptography, data encryption, cipher, error control codes, the Reed-Muller forms, etc. [2, 6, 11, 14, 17, 18, 19, 22, 24] Such investigations are continuously developed and mostly concentrate on the construction of fully defined functions. On the other hand, in modern digital circuits analysis and synthesis, very often incompletely defined Boolean functions are used [15]. For this reason, it is very important how to minimize and build appropriate forms of function.

For a given partially defined Boolean function it is hard to estimate whether the function can be realized as affine, especially for large functions. These troubles can be overcome by many techniques, especially by spectral techniques.

Techniques based on the Walsh transform allow to investigate Boolean functions in many areas such as classification, disjoint decomposition, multiplexer and threshold logic synthesis, state assignment, testing and evaluation of logic complexity [4, 10, 16, 18]. Search of linearity is also profitable for some Boolean functions implementations. The affine Boolean functions are also intensively investigated in cryptography and data encryption areas.

The described method allows to check whether a partially defined Boolean function can be realized in an affine form.

2 BASIC DEFINITIONS

Let $\mathbf{B}$ be a set of elements that forms a Boolean algebra. The simplest one is $\mathbf{B} = \{0, 1\}$. A Boolean function f is a mapping $f : \mathbf{B}^n \to \mathbf{B}$. This domain is the set of 2^n binary vectors $(0, 0, \ldots, 0), (0, 0, \ldots, 1), \ldots, (1, 1, \ldots, 1)$. Thus, the Boolean function is a set of ordered pairs in which the first element is a binary input vector and the second element is the constant 0 or 1. If domain $\mathbf{D}$ of Boolean function f is $\mathbf{B}^n$ then f is a completely specified function. If $\mathbf{D} \subset \mathbf{B}^n$ then f is incompletely defined. If some values of the function f belong to set $\mathbf{B}^n \backslash \mathbf{D}$ then f is incompletely defined. The points where the value for f is not assigned are called *don't cares.* The binary vectors for which function is undefined will be called DC cubes. The power of the set DC will be denoted as $card(\mathrm{DC}) = d$. The undefined values of the Boolean function will be denoted by the symbol $'-'$. Hence, an n -variable incompletely specified switching function is a mapping $f : \mathbf{B}^n \to \mathbf{B} \cup \{-\}$. The Boolean function f can be specified by enumerating its values at all the decimal indices, which can be conveniently represented by truth-vector $\mathbf{Y}_f = [y_0, y_1, \dots, y_{2^n-1}]$ of f.

Definition 1. The true (false) set of the Boolean function, denoted by $T_f(F_f)$ is a collection of all true (false) vectors of f, i.e. $T_f = \{x \in \{0,1\}^n : f(x) = 1\}$ – it is the so called set of ON cubes, and $F_f = \{x \in \{0,1\}^n : f(x) = 0\}$, it is the so called set of OFF cubes.

Definition 2. The Boolean function $f_k(x_1, x_2, \dots, x_n)$ of $n-$ variables is called affine if it can be represented as $f_k(x) = a_1x_1 \oplus a_2x_2 \dots \oplus a_nx_n \oplus c$, where $a_j, c \in \{0,1\}$ and $k = c + \sum_{i=1}^{n} a_i 2^i$. In particular, if $c = 0$ then f_k is called a linear function.

Definition 3. The scalar product of the two vectors $\mathbf{X} = [x_0, x_1, \dots, x_m]$ and $\mathbf{Y} = [y_0, y_1, \dots, y_m]$ is a number and is calculated as $< \mathbf{X}, \mathbf{Y} >= x_0y_0 + x_1y_1 + \dots + x_my_m$.

The Boolean function $f(x_1, x_2, \dots, x_n)$ given by the binary truth-vector $\mathbf{Y}_f = [y_0, y_1, \dots, y_{2^n-1}]^T$ can be transformed from a Boolean domain $\{0,1\}$ into the spectral domain by the linear transformation $\mathbf{H} \cdot \mathbf{Y}_f = \mathbf{S}$, where $\mathbf{H}$ is a $2^n \times 2^n$ transform matrix, and $\mathbf{S} = [s_0, s_1, \dots, s_{2^n-1}]^T$ is the vector of spectral coefficients called spectrum of f [9, 10, 8]. It is the Fourier transform of the Boolean function. Usually such a transform is also called Walsh-Hadamard Transform and the decomposition of the Hadamard matrix $\mathbf{H}$ can be written as

$$\mathbf{H}_N = \prod_{i=1}^{n} (\mathbf{I}_{2^{n-i}} \otimes \mathbf{P}_{2^i}) \qquad (1)$$

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad \mathbf{P}_2 = \mathbf{H}_2, \mathbf{P}_N = \begin{bmatrix} \mathbf{I}_{N/2} & \mathbf{I}_{N/2} \\ \mathbf{I}_{N/2} & -\mathbf{I}_{N/2} \end{bmatrix}$$

where $\otimes$ denotes the Kronecker product, $N = 2^n$, $\mathbf{I}_N$ is the identity matrix of size N and:

For instance, if $n = 2$, then from Equation (1) the Hadamard matrix has the form

$$\mathbf{H}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Each row of the matrix $\mathbf{H}_N$ includes a discrete Walsh sequence $wal(w,t)$. In this notation, $w = 1, \dots, 2^n$ identifies the index of the Walsh function, and $t = 1, \dots, 2^n$ stands for a discrete point of the function determination interval.

It follows from Definition 1 that a number of undefined points of f can be calculated from the formula $d = 2^n - [card(T_f) + card(F_f)]$. If $d = 0$ then the Boolean function f is fully defined.

The coefficient s_0 is directly related to the number of minterms for which the Boolean function f has the value 1. If the number of true minterms will be denoted by a, then $s_0 = 2^n - 2a$. Additionally, for a partially defined Boolean function we have $s_0 = 2^n - 2a - d$ [4].

3 SPECTRAL IDENTIFICATION OF AN AFFINE BOOLEAN FUNCTION

In many practical problems Boolean functions are specified in an incomplete form. In such cases truth vector $\mathbf{Y}_f$ of f has the values $\{0, 1, -\}$, where the symbol "$-$" denotes *don't care* minterms. In order to obtain spectral coefficients $s \in \mathbf{S}$, elements of $\mathbf{Y}_f$ are re-coded, according to the formula $\{0, 1, -\} \to \{1, -1, 0\}$.

Theorem 1. [3, 18] The affine Boolean function f is characterized by the unique Walsh-Hadamard spectrum distribution

$$s_x = \begin{cases} +2^n & \text{for} \quad x = k/2 \quad\quad \Leftrightarrow c = 0 \\ -2^n & \text{for} \quad x = (k-1)/2 \quad \Leftrightarrow c = 1 \\ 0 & \quad\quad \text{otherwise} \end{cases} \tag{2}$$

where $k = c + \sum_{i=1}^{n} a_i 2^i$, $a_j, c \in GF(2)$ has the same meaning as in Definition 2 and $x = 0, 1, \ldots, 2^n - 1$.

Proof. It is known directly from the definition of the Walsh functions that they form a complete orthogonal system [1]. The mutual orthogonality rows of the Hadamard matrix satisfy the condition

$$\sum_{t=0}^{2^n-1} wal(i,t) \cdot wal(j,t) = \begin{cases} 2^n & \text{for} \quad i = j \\ 0 & \text{for} \quad i \neq j \end{cases} \tag{3}$$

$$\sum_{t=0}^{2^n-1} wal(i,t) = \begin{cases} 2^n & \text{for} \quad i = 0 \\ 0 & \text{for} \quad i \neq 0. \end{cases} \tag{4}$$

Rows of the matrix $\mathbf{H}_N$ include the truth vectors of all n variable Boolean functions [18]. It follows from Definition 2 that if function f is linear, then its complement has the form $1 \oplus f$ and it is an affine Boolean function. In the Walsh-Hadamard Transform, elements of the truth vector of the Boolean function are coded according to the formula $\{0, 1\} \to \{1, -1\}$. In this case, if the truth vector of the Boolean function is denoted by $\mathbf{Y}_f$, then its complementary version has the form $(-1)\mathbf{Y}_f$. If this vector represents the linear Boolean function, then its coordinates are the same as (or complement to) one of the rows of matrix $\mathbf{H}_N$. Hence, using (3), (4), we obtain (2). □

Theorem 2. A given Boolean function f, with n variables, undefined at one point only, can be affine if f is characterized by the Walsh-Hadamard spectrum distribution

$$s_x = \begin{cases} +1 & \text{for } x = 0 & \rightarrow \text{all undefined points have value 1} \\ -1 & \text{for } x = 0 & \rightarrow \text{all undefined points have value 0} \\ +(2^n-1) \text{ for} & x = k/2 & \rightarrow c = 0 \\ -(2^n-1) \text{ for} & x = (k-1)/2 & \rightarrow c = 1 \\ \pm 1 & \text{otherwise} & \end{cases} \tag{5}$$

where k and c have the same meaning as in Definitions 2 and $x = 0, 1, \ldots, 2^n - 1$.

Proof. It is well known that an affine Boolean function is balanced. Because coordinates of its truth vector are coded as $\{0,1\} \rightarrow \{1,-1\}$, vector $\mathbf{Y}_f$ of function f includes 2^{n-1} elements of $+1$, and 2^{n-1} elements of (-1). Additionally, from the properties of the matrix $\mathbf{H}_N$ follows that the first row of such a matrix consists of 1's only. In consequence, $s_0 =< \mathbf{1}, \mathbf{Y}_f >$. Assume that in vector $\mathbf{Y}_f$ one coordinate (say -1) will be replaced by $' - '$, then this point is coded by the value of 0. Hence, $s_0 =< \mathbf{1}, \mathbf{Y}_f > = 2^{n-1} \times 1 + (2^{n-1} - 1) \times (-1) + 0 \times (-1) = 1$. Similarly, if one coordinate with the value of 1 will be replaced by $' - '$, then $s_0 =< \mathbf{1}, \mathbf{Y}_f > = 2^{n-1} \times (-1) + (2^{n-1} - 1) \times 1 + 0 \times 1 = -1$.

If the incompletely defined Boolean function f can be extended to a linear (affine) form, coordinates of some k^{th} row of $\mathbf{H}_N$ and coordinates of $\mathbf{Y}_f$ are the same (or complement), except for the coordinate $' - '$ in $\mathbf{Y}_f$. For this reason, $2^n - 1$ appropriate coordinates of $\mathbf{H}_N$ and $\mathbf{Y}_f$ are the same ($c = 0$) or complement ($c = 1$). Because the first row of $\mathbf{H}_N$ consists of elements ± 1, then $s_i = (2^n - 1) \times 1 = 2^n - 1$ or $s_i = (2^n - 1) \times (-1) = -(2^n - 1)$. □

Proposition 1. Let f be a partial Boolean function which is undefined at d points. Function f can be extended up to an affine form and f is given by the vector $[y_0, y_1, \ldots, y_{2^n-1}]$, where $y_i \in \{+1, -1, 0\}$. Then:

a) If all $' - '$ points are located in set T_f, then $s_0 = +d$.

b) If all $' - '$ points are located in set F_f, then $s_0 = -d$.

c) If all $' - '$ points are located in both sets T_f and F_f, then $s_0 \neq d$ and for some $i = 1, \ldots, 2^n - 1$, $s_i = \pm(2^n - d)$.

Proof. The proof follows from Theorem 2. The affine Boolean function f is balanced and is represented by the vector $\mathbf{Y}_f = [y_0, y_1, \ldots, y_{2^n-1}]$. This vector consists of 2^{n-1} elements of $+1$, and 2^{n-1} elements of (-1). Let us assume that $\mathbf{Y}_f$ includes d elements, coded by the value 0 (d undefined points). If these points are located in set T_f, then $s_0 =< \mathbf{1}, \mathbf{Y}_f > = (2^{n-1} \times 1) + (2^{n-1} - d) \times (-1) = +d$. If these points are located in set F_f, then $s_0 =< \mathbf{1}, \mathbf{Y}_f > = (2^{n-1} - d) \times 1 + 2^{n-1} \times (-1) = -d$.

If function f can be extended to an affine form, then for some k $[y_0, y_1, \ldots, y_{2^n-1}] = wal(k,t)$, for $t = 0, \ldots, 2^n - 1$. According to Theorem 1, if f is fully defined ($d = 0$), then $s_k = \pm 2^n$. If $\mathbf{Y}_f$ has freely placed $d = p_0 + p_1$ undefined points, there are $(2^{n-1} - p_0)$ elements of 1 and $(2^{n-1} - p_1)$ elements of (-1) in both $\mathbf{Y}_f$

and *wal*(k,t) vectors. These elements are located in the same positions in both mentioned vectors or elements of $\mathbf{Y}_f$ are complement to *wal*(k,t). Hence, $s_k =< wal(k,t), \mathbf{Y}_f > = (2^{n-1} - p_0) \times 1 - (2^{n-1} - p_1) \times (-1) = 2^n - (p_0 + p_1) = 2^n - d$ or $s_k = (2^{n-1} - p_0) \times (-1) - (2^{n-1} - p_1) \times (1) = -2^n + d$. □

The incompletely defined Boolean functions were considered among other things in [4], where some spectral properties of such functions were performed but spectrum distribution of affine functions was not presented.

Example 1. Table 1 includes the description of some Boolean functions. Function $f_1(x)$ is fully defined. This function is linear. The remaining functions are incompletely defined and each of them has a different number of undefined points d.

$x_1x_2x_3$	$f_1(x)$ $(d=0)$	s_x	$f_2(x)$ $(d=1)$	s_x	$f_3(x)$ $(d=2)$	s_x	$f_4(x)$ $(d=2)$	s_x	$f_5(x)$ $(d=2)$	s_x	$f_6(x)$ $(d=3)$	s_x	$f_7(x) = 1 \oplus f_6(x)$ $(d=3)$	s_x
000	0	0	0	−1	0	−2	0	2	0	0	0	−1	1	1
001	1	0	1	−1	1	0	–	−2	–	−2	–	−1	–	1
010	0	0	–	1	–	0	0	0	–	2	–	1	–	−1
011	1	0	1	1	1	2	–	0	1	0	1	1	0	−1
100	1	0	1	−1	1	0	1	2	1	0	1	1	0	−1
101	0	8	0	7	–	6	0	6	0	6	–	5	–	−5
110	1	0	1	1	1	2	1	0	1	2	1	3	0	−3
111	0	0	0	1	0	0	0	0	0	0	0	−1	1	1

Table 1. The Boolean functions and their spectra

It follows from Theorem 2 that functions $f_i(x)$, $i = 2, \ldots, 7$ can be extended up to an affine function $f_1(x)$.

4 OTHER METHODS OF SPECTRA CALCULATION

Spectral methods have been widely used in many domains for a long time. The Discrete Fourier Transform (DFT) is a specific kind of Fourier transform, used in Fourier analysis. DFT can be computed efficiently in practice using a Fast Fourier Transform (FFT) algorithm. The use of spectral methods dates back to the early 1960s. Nowadays, in modern digital analysis and synthesis spectral techniques are often used, and are comparatively simple and very well known. Additionally, many methods and algorithms can be demonstrated where spectral coefficients are effectively calculated. The classical approach to computing the Walsh-Hadamard spectrum is based on the truth table of a Boolean function. The most effective truth table-based algorithm is the Fast Walsh-Hadamard Transform [1, 9, 10]. Unfortunately, the main disadvantage of such a method is that it cannot be used for logic functions with large numbers of variables, because the main limiting factor of spectral methods in processing of switching functions is their calculation complexity [6]. For example, the time and space complexities of the FFT-like algorithms are $O(n2^n)$ and $O(2^n)$, respectively, for Boolean functions of n variables. One way to overcome the above difficulties is to use the Spectral Decision Diagrams (SDDs) to compute the Walsh spectral coefficients [6, 26, 27, 29, 25]. These diagrams are very convenient data structures for the majority of discrete functions [28]. As said before, computations even with the fast transforms can be difficult, because the truth-table of Boolean functions grows expotentially with n. For this reason, methods based on SDDs are preferred.

The mentioned methods for a completely as well as incompletely defined Boolean function can be used. It can be observed that besides of SDDs technique, fast transforms based on butterfly charts are still applied. It is because Decision Diagrams structures, although often used, are difficult to hardware realization unlike spectral algorithms which are very efficient realized as circuit applications. Additionally, if a function is fully defined, FFT algorithms have the same computational complexity as the SDD techniques [6, 26]. The advantage of SDD techniques can be observed for incompletely defined Boolean functions, but in such cases all undefined points of a function are automatically replaced by 0(1) value. Because the butterfly and SDDs-based algorithms are well known today, they will not be presented here.

Recently, a method to compute the Walsh-Hadamard spectrum directly from Sum of Products (SOP) representation has been proposed. However, many practical logic functions cannot be represented in an SOP form, because the numbers of such products can be too large [6].

Obviously, after the minimizing procedures the function has another Boolean and spectral representation. Such a form is of course very compact but at this juncture the information whether a function could be affine is lost.

Example 2. Let a Boolean function be described as $f^v = x_1 \oplus x_2 \oplus x_3$ The truth vector of such a function has the form $\mathbf{Y}_{f^v} = [0,1,1,0,1,0,0,1]$. On the basis of vector $\mathbf{Y}_{f^v}$, by means of ESPRESSO algorithm [12] SOP form is generated $f_1^v = \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + x_1\overline{x}_2\overline{x}_3 + x_1x_2x_3$. Modern and very efficient algorithm EXORCISM4 [13], which generates the Reed-Muller forms, gives ESOP form $f_2^v = \overline{x}_1 \oplus x_2 \oplus \overline{x}_3$. It follows from this example that f^v representation is the best and can be realized by means of one EXOR gate.

The Spectral Decision Diagram representations permit to calculate spectral coefficients via graph-based algorithms. It is a very fast calculation method. Classical SDD and reduced SDD (RSDD) of function f^v are shown in Figures 1 a) and 1 b). The function from Example 2 has the following Walsh-Hadamard spectrum $\mathbf{S}_{f^v} = [0, 0, 0, 0, 0, 0, 0, 8]$. Hence, according to Theorem 1 the analyzed function is linear.

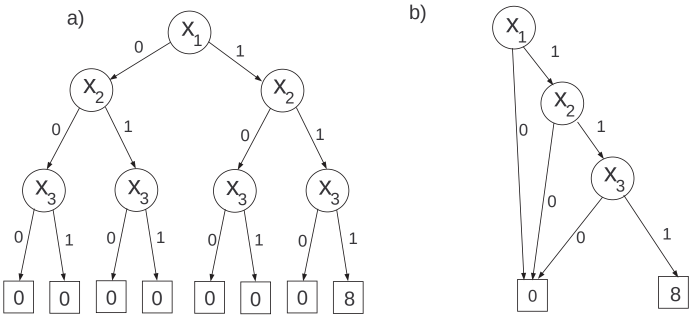

Fig. 1. Spectrum of a fully defined Boolean function: a) Spectral Decision Diagram b) reduced SDD

If vector $\mathbf{Y}_{f^v}$ will be modified as follows $\mathbf{Y}_{f^*} = [0, -, 1, 0, 1, -, 0, 1]$, then EXPRESSO gives an SOP form $f_1^* = x_1\overline{x}_2 + \overline{x}_1 x_2 \overline{x}_3 + x_1 x_3$ and EXORCISM4 gives an ESOP form $f_2^* = x_1 \oplus x_1\overline{x}_2 x_3 \oplus x_2\overline{x}_3$. Now, the spectrum of the new function f^* depends on the extension method. For this reason we obtain the different spectra: $\mathbf{S}_{f_1^*} = [0, 0, 0, 0, 4, -4, 4, 4]$ and $\mathbf{S}_{f_2^*} = [2, -2, 2, -2, 2, -2, 2, 6]$, respectively. It can be observed that all *don't care* points were replaced differently. In EXPERESSO algorithm these points were replaced by elements of $\{0, 1\}$; while in EXORCISM4 algorithm, such points are replaced by the value 0. Additionally, it can be easily checked that $f_{_1}^* \neq f_{_2}^*$ and both functions are not linear now.

The final form of SDD's for the non-linear function f_2^* has been depicted in Figures 2 a) and 2 b). Thus, instead of a simple linear function f^v, the another function f_2^* was generated.

Taking into account the mentioned considerations, for the Boolean functions which have $n \leq 20$ variables both FFT-like and SDDs methods can be applied. For a large Boolean function ($n > 20$) the Spectral Decision Diagrams are preferred. It can be observed that for the fully defined Boolean functions is also possible to obtain so-called pruned Walsh coefficients [7]. In such methods based on Decision Diagrams, only selected spectral coefficients can be generated without calculating the complete spectra.

In this work, for spectral coefficients calculation the method described in [4] has been used. In this approach a complete set of spectral coefficients is generated. The method is especially preferred in cases when a Boolean function has a lot of DC-cubes. By means of this method the same spectrum as in FFT algorithm is obtained

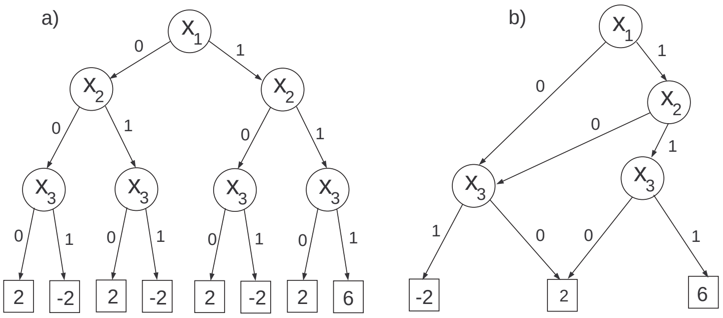

Fig. 2. a) The SDD of the Boolean function f_2^* from Example 2, b) its reduced SDD form

because optimizing procedures, like in EXPRESSO and EXORCISM4 algorithms, are not used. Spectrum is performed on the basis of arrays of disjoint ON- and DC-cubes. The information about OFF-cubes is not necessary, therefore such a representation is very compact. Disjoint cubes can be directly obtained, for example, from a well known ESPRESSO algorithm (with switch -Ddisjoint).

Example 3. A given Boolean function f can be represented by means of ON and DC cubes as follows:

x_1, x_2, x_3 type of cube	$s_{j,0}$	$s_{j,1}$	$s_{j,2}$	$s_{j,3}$	$s_{j,4}$	$s_{j,5}$	$s_{j,6}$	$s_{j,7}$	no. of cube
0 - 1 ON	4	-4	0	0	4	4	0	0	$j = 1$
1 - 0 ON	4	4	0	0	-4	4	0	0	.
1 0 1 DC	3	1	-1	1	1	-1	1	-1	.
0 1 0 DC	3	-1	1	1	-1	-1	1	1	$j = w + k$
Total spectrum $s^{tot}_{i=0,\ldots,2^n-1} =$	-2	0	0	2	0	6	2	0	

Table 2. The cubes representation of a given Boolean function and its spectrum

The spectral coefficients are calculated in the table rows, separately for ON and DC cubes. The spectral coefficients for ON cubes are calculated from the formulas $s^{ON}_{1,0} = 2^n - 2 \cdot 2^p$, $s^{ON}_{j,i} = \pm 2 \cdot 2^p$, $j = 1, \ldots, k$, $i = 1, \ldots, 2^n - 1$. The spectral coefficients for DC cubes are calculated from the formulas $s^{DC}_{j,0} = 2^{n-1} - 2^p$, $j = k+1, \ldots, w+k$, $i = 1, \ldots, 2^n - 1$, $s^{DC}_{j,i} = \pm 2^p$, where p is the number of symbols $' - '$ in the given cube, and the remaining variables have the same meaning as previously.

The final coefficient s_0^{tot} is calculated from the formula $s_0^{tot} = (\sum_{j=1}^{w+k} s_{j,0}) - (k - 1) \times 2^n - w \times 2^{n-1}$, where k is the number of the disjoint ON cubes, w is the number of the disjoint DC cubes, and n is the number of variables of the Boolean function. The remaining coefficients are calculated from the general formula $s_i^{tot} = (\sum_{j=1}^{w+k} s_{j,i})$. More details about spectra calculations can be found in [4, 5]. The representation of the cubes presented in Table 1 is equivalent to vector $\mathbf{Y}_f = [0, 1, -, 1, 1, -, 1, 0]$ of the function f. The same spectral coefficients will be obtained in case when the classical Walsh-Hadamard transform will be applied (Equation (1)).

The methods of spectra calculation are not presented here exhaustively – such methods are described in many works mentioned in this paper. Spectral coefficients analysis is more interesting because from coefficient values and their distribution some important properties of the Boolean function can be recognized.

5 ALGORITHM TO SEARCH THE AFFINE EXTENSION OF AN INCOMPLETELY DEFINED BOOLEAN FUNCTION

Taking into account Theorems 1 and 2 recurrence method of searching of an affine function can be presented. This method based on truth vector $\mathbf{Y}_f$ (or cubes representation) of an incompletely defined Boolean function. If an affine representation exists, an appropriate form will always be found. The fundamental part of the algorithm is rejecting such incompletely defined functions, for which it is not possible to construct affine functions. Because for any n-variable Boolean function we obtain $2^{(2^n)}$ different Boolean functions and only 2^{n+1} of them are affine, fast estimation of an incomplete function seems to be important. For small functions, recognition of linearity of a Boolean function is not difficult. Some troubles are performed for large and weakly defined functions, because simple extension to the full form can not always be recognized.

The main steps of the proposed algorithm are presented below. Every step is described in detail.

Input data of the algorithm The incompletely defined Boolean function $f(x_1, x_2, \ldots, x_n)$ represented by vector Y_f (or cubes) and number of its undefined points d

Output data of the algorithm The set of all affine functions which can be generated on the basis of $\mathbf{Y}_f$

Step 1 Compute the spectral coefficients $s_i \in \mathbf{S}$ of function f, $i = 1, \ldots, 2^n - 1$.

Step 2 If $s_0 = \pm d$, then f can be directly extended to an affine form, go to *Step 5*. Otherwise,

Step 3 if the condition $\max\{|s_1|, |s_2|, \ldots, |s_{2^n-1}|\} = \pm(2^n - d)$ is fulfilled, then f can be realized in an affine form, go to *Step 4*, otherwise go to *Step 5*.

Step 4 Call the procedure of function f completion on the basis of don't care places (subroutine EXTENSION).

Step 5 End of algorithm.

In the above mentioned algorithm, the subroutine EXTENSION is used, and it is an integral part of the presented algorithm. This part of the algorithm is used in the cases when simple extension of f is not possible because $s_0 \neq d$ but the condition $s_i = \pm(2^n - d)$ is fulfilled. If f can be extended up to the affine form, then truth-vector of f can be written as $\mathbf{Y}_f = [ab]$, where $a = b$ or $a = \overline{b}$ or $\overline{a} = b$ or $\overline{a} = \overline{b}$and a, b are subvectors of length 2^{n-1}.

The subroutine EXTENSION is built as a recurrence function. The main parts of the pseudo code can be described as follows:

```
EXTENSION($Y_f$)
If Number_of_bits($Y_f$) > 1
    Divide_$Y_f$_into_two_substr: Left_substr,Right_substr
case Number_of_DC_bits(Left_substr) = 0
        EXTENSION(Right_substr)
        For_all_auxiliary_vectors_(WP)_to_do:
        {
            Number:=Compare(WP(i), Left_substr)
            Expand(WP(i), Number)
        }
    case Number_of_DC_bits(Right_substr) = 0
        EXTENSION(Left_substr)
        For_all_auxiliary_vectors_(WP)_to_do:
        {
            Number:=Compare(WP(i), Right_substr)
            Expand(WP(i), Number)
        }
    case Number_of_DC_bits(Right_substr) >= Number_of_bits(Left_substr)
        EXTENSION(Left_substr)
        For_all_auxiliary_vectors_(WP)_to_do:
        {
            Number:=Compare(WP(i), Right_substr)
            Expand(WP(i), Number)
        }
    case Number_of_DC_bits(Left_substr) > Number_of_bits(Right_substr)
        EXTENSION(Right_substr)
        For_all_auxiliary_vectors_(WP)_to_do:
        {
            Number:=Compare(WP(i), Left_substr)
            Expand(WP(i), Number)
        }
    endcase
else
    WP(0):=[0]
    WP(1):=[1]
end
```

The auxiliary vector $WP(i)$ and *Left_substr* or *Right_substr* of the vector $\mathbf{Y}_f$ as parameters of the function Compare are used. The either *Left_substr* or *Right_substr* are the sub-vectors which include *don't care* states. Completion of the sub_vectors is conducted by means of the appropriate bits comparison between $WP(i)$ and one from sub_vectors *Left_substr* (*Right_substr*). The function Compare returns values which are presented in Table 3.

-1	*Left_substr* or *Right_subst* can not be completed by the auxiliary vector $WP(i)$.
1	*Left_substr* or *Right_subst* can be completed by the auxiliary vector $WP(i)$.
2	*Left_substr* or *Right_subst* can be completed by negation of vector $WP(i)$.
3	*Left_substr* or *Right_subst* can be completed by negation of the vector $WP(i)$ or else by $WP(i)$.

Table 3. Values returned by the function Compare

The auxiliary vector $WP(i)$ and the value *Number* are used as parameters of the function Expand. The value of the parameter *Number* is known because that value is returned by the function Compare. Depending on value of the parameter *Number*, appropriate operations are executed (cf. Table 4).

Number	Description of operations
-1	Delete the vector $WP(i)$ from the list of auxiliary vectors.
1	Execute concatenation $WP(i) := WP(i) \bowtie WP(i) + 1$.
2	Depending on which part of the vector $\mathbf{Y}_f$ will be analyzed, execute $WP(i) := WP(i) \bowtie neg[WP(i)]$ or $WP(i) := neg[WP(i)] \bowtie WP(i)$.
3	Vector $WP(i)$ is stored by substitution $WT := WP(i)$. $WP(i) := WP(i) \bowtie WP(i)$ and depend on which part of vector $\mathbf{Y}_f$ will be analyzed, execute $WP(i+1) := WT \bowtie neg[WT]$ or $WP(i+1) := neg[WT] \bowtie WT$. Include the vector $WP(i)$ to the list of auxiliary vectors.

where $\mathbf{A} = [a_1, a_2, \dots, a_k]$, $\mathbf{B} = [b_1, b_2, \dots, b_k]$ – the binary vectors, $\mathbf{A} \bowtie \mathbf{B}$ – concatenation of the vectors $\mathbf{A}$ and $\mathbf{B}$, $neg(\mathbf{A})$ – bits negation of vector $\mathbf{A}$.

Table 4. Operations of the Expand function depend on the *Number* value

The spectral test allows us to check whether from an incompletely defined Boolean function affine functions can be generated. In this way searching of affine functions will be significantly improved. That feature is very important because instead of all Boolean functions (for a given n) we have only a few affine functions. Brute force searching in these cases is not efficient.

For instance if truth vector of a given Boolean function can be written as $\mathbf{Y}_f = [1{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}1{-}{-}{-}{-}{-}{-}{-}{-}0{-}{-}{-}{-}{-}{-}]$, ($n = 5$) then on the basis of algorithm, the 8 affine functions will be generated. For vector $\mathbf{Y}_f$ = [01–00110–001100110–1100101–00110] we obtain only one linear function ($s_{x=27} = +32$) which can be represented by means of the binary vector [01100110100110011001100101100 110].

Complexity time of the classical Fast Walsh Transform (FWT) is equal to $O(2^n \log 2^n)$ [1]. Unfortunately such a method can be used for small Boolean functions (say $n < 20$). For large functions, the Walsh matrices of dimension $2^n \times 2^n$ are very inconvenient. In the method [5], time complexity decreases according to the number of literals (l) as well as to the number of cubes (c) in function f. The authors state that upper bound time for spectral coefficient calculation is estimated by the function $t(l, c) = 2^{l-12} \times c$, where c is a constant depending on a computer. It is complexity for determination of 2^n spectral coefficients. In our case, this complexity can be strongly decreased because in many cases an incomplete Boolean function can be directly extended if the coefficient s_0 has appropriate value – in this case additional spectra calculation can be passed over.

It was explained above that SDD approach for an incompletely defined Boolean function is inconvenient, because before computations, all *don't care* places are arbitrarily replaced by the value 0 or 1. For the fully defined Boolean functions time complexity of SDDs method is also equal to $O(2^n \log 2^n)$ [26]. The variable n has the same meaning as in Definition 2

In the next investigation, the time of extension to all affine forms has been determined. It was done by means of the long-time experiments where 10 000 randomly selected Boolean functions were used, with different literals ($l = 1 \ldots 40$) and cubes ($c = 1 \ldots 40$) were applied. During tests, time of the Boolean function extension was measured for a different number of literals and cubes. For appropriate n, time values were averaged. Figure 3 presents an average time of affine function searching for two cases.

Computation time was registered for cases when it was greater than 1 ms. In the first case spectrum of a Boolean function was calculated by means of the Fast Walsh-Hadamard Transform (FWHT). In the second case the spectrum is based on algorithm [5]. The first method of calculations was applied for the number of function arguments $n \leq 20$. For larger functions, matrices (1) are too large for computer programmes. The second method of spectrum calculation is very convenient even for large Boolean functions, especially when a function has a small number of disjoint cubes. The method of a Boolean function searching was tested by means of a PC, where the Windows operating system was installed with Celeron 2.4 GHz and 256 MB DDR RAM. All times are given in CPU milliseconds.

6 CONCLUSIONS

The proposed spectral method allows to obtain fast information about linearity of the analyzed function. The method of the spectral coefficients analysis can be used for any incompletely specified Boolean functions.

In the paper, the relationship between the Walsh-Hadamard spectrum and the incompletely defined Boolean functions has been discussed.

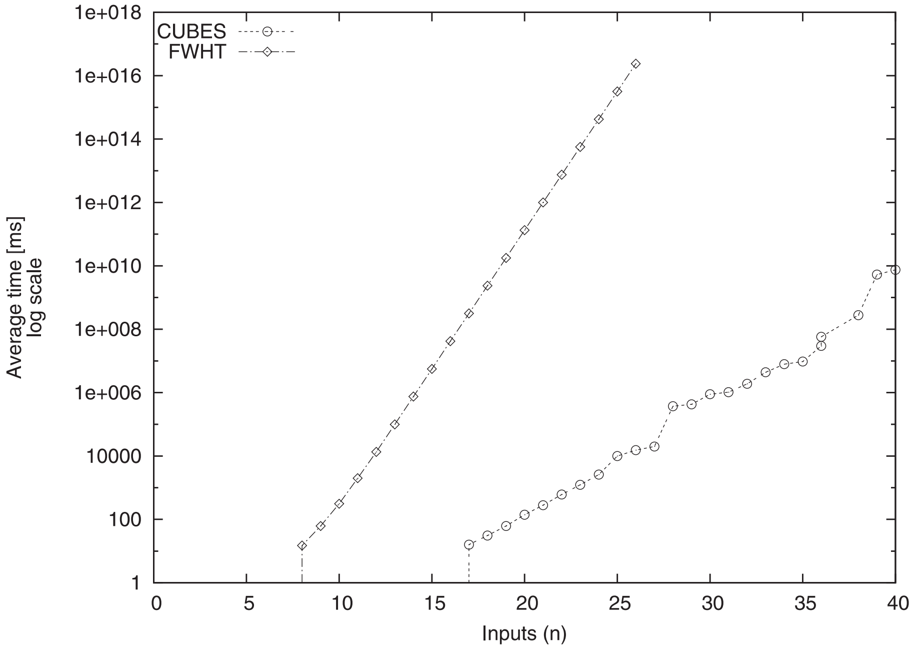

Fig. 3. Complexity time of the extension of a weakly defined Boolean function to its full affine form

The presented theorems and propositions show a new way to find affine Boolean functions, from their incomplete description. Spectral coefficients can be calculated with the aid of different methods. Fortunately, such methods are nowadays well described and known. A short review of spectra calculation methods has also been stated. If an incompletely defined Boolean function can be extended to an affine form, then by means of a simple algorithm their full defined form can be generated.

REFERENCES

[1] Ahmed, N.—Rao, K. R.: Orthogonal Transforms for Digital Signal Processing. Springer-Verlag Berlin 1975.

[2] Canteaut, A.—Carlet, C.—Charpin, P.—Fontaine, C.: On Cryptographic Properties of the Cosets of $R(1,m)$. IEEE Trans. Inform. Theory. Vol. 47, 2001, No. 4, pp. 1494–1513.

[3] Falkowski, B. J.—Porwik, P.: Evaluation of Nonlinearity in Boolean Functions by Extended Walsh-Hadamard Transform. 2^{nd} Int. Conf. on Information Communications and Signal Processing ICISC Singapore 1999, paper 2B2.2, pp. 1–4.

[4] Falkowski B. J.—Schafer, I.—Perkowski, M. A.: Effective Computer Methods for the Calculation of Rademacher-Walsh Spectrum for Completely and Incom-

pletely Specied Boolean Functions. IEE Trans. on Computer-Aided Design, Vol. 11, 1992, No. 10, pp. 1207–1226.

[5] Falkowski, B. J.—Shäfer, I.—Perkowski M.: A Fast Computer Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions. Proc. 33$^{\text{rd}}$ Midwest Symp. on Circuits and Systems, Calgary, Canada 1990, pp. 1119–1122.

[6] Fujita, M.—Yang, J.: Fast Spectrum Computation for Logic Functions using Binary Decision Diagrams. Proc. Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Japan, 1995, pp. 275–278.

[7] Jankovic, D.—Stankovic, R.—Drechsler, R.: Decision Diagram Method for Calculation of Pruned Walsh Transform. IEEE Trans. on Comp., Vol. 50, 2001, No. 2, pp. 147–157.

[8] Astola, J. T.—Stankovic, R.: Fundamentals of Switching Theory and Logic Design. Springer Verlag 2006.

[9] Hurst, S. L.—Miller, D. M.—Muzio, J. C.: Spectral Techniques in Digital Logic. Academic Press, London 1985.

[10] Karpovsky, M. G.: Finite Orthogonal Series in the Design of Digital Devices. John Wiley, New York 1976.

[11] Maitra, S.—Sarkar, P.: Cryptographically Significant Boolean Functions with Five Valued Spectra. Theoretical Computer Science 2002, No. 276, pp. 133–146.

[12] de Micheli, G.: Synthesis and Optimization of Digital Circuits. McGraw-Hill, Boston 1994.

[13] Mishchenko, A.—Perkowski, M.: Fast Heuristic Minimization of Exclusive-Sum-of-Products. Proc. of Reed-Muller Workshop, Starkville, Mississippi 2001, pp. 242–250.

[14] Mister, S.—Adams, C.: Practical S-Box Design. Workshop on Selected Areas in Cryptography (SAC 96), Queen's University Kingston, Ontario, Canada, 1996, pp. 61–76.

[15] Matsuura, M.—Sasao, T.: Representation of Incompletely Specified Switching Functions Using Pseudo-Kronecker Decision Diagrams. Int. Workshop on App. of the Reed-Muller Expansion in Circuit Desig, Starkville, Mississippi (USA) 2001, pp. 27–33.

[16] Porwik, P.—Falkowski, B. J.: Informatics Properties of the Walsh Transform. 2$^{\text{nd}}$ Int. Conf. on Information Communications and Signal Processing (ICISC 99), Singapore, paper 2B2.4, 1999, pp. 1–5.

[17] Porwik, P.: Efficient Calculation of the Reed-Muller Forms by Means of the Walsh Spectrum. Int. Journal of Applied Mathematics and Computer Science, Vol. 12, 2002, No. 4, pp. 571–579.

[18] Porwik, P.: The Spectral Test of the Boolean Function Linearity. Journal of Applied Mathematics and Computer Science, Vol. 13, 2003, No. 4, pp. 567–575.

[19] Porwik, P.: Efficient Algorithm of Affine Form Searching for Weakly Specified Boolean Function. Fundamenta Informaticae, Vol. 77, 2007, No. 3, IOS Press, Amsterdam, pp. 277–291.

[20] Quing, M.—Min, Y.—Huanguo, Z.—Jingsong, C.: A Novel Algorithm Enumerating Bent Functions. Discrete Mathematics, Vol. 38, 2007, No. 23, pp. 5576–5584.

[21] Sasao, T.—Besslich, P.: On the Complexity of Mod-2 Sum PLA.s. IEEE Trans. on Comp. Vol. 39, 1990, No. 2, pp. 262–266.

[22] Sasao, T.: Logic Synthesis and Optimization. Kluwer Academic Publisher, Dordrecht (Holland) 1993.

[23] Sasao, T.: Representation of Logic Functions using EXOR Operators. Workshop on Applications of the Read-Muller Expansion in Circuit Design. Makuhari, Japan, 1995, pp. 308–313.

[24] Seberry, J.—Zhang, X. M.: Construction of Bent Function from Two Known Bent Functions. Australasian Journal of Combinatories, Vol. 9, 1994, pp. 21–34.

[25] Somenzi, F.: BDD Package. CUDD v.2.3.0. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`.

[26] Stanković, R.—Astola, J. B.: Spectral Interpretation of Decision Diagrams. Springer Verlag 2003.

[27] Thornton, M.—Drechsler, R.: Spectral Decision Diagrams Using Graph Transformations. Int. Conf. Design, Automation and Test in Europe, Munich (Germany), 2001, pp. 713–719.

[28] Wegener, I.: Branching Programs and Binary Decision Diagrams. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia (USA) 2000.

[29] Yang, C.–Ciesielski, M.: BDS: A BDD-Based Logic Optimization System. IEEE Trans. CAD, Vol. 21, 200, No. 7, pp. 866–876.

Piotr Porwik is a Professor of computer science in the Computer System Department, Computer Science and Materials Science Faculty, Katowice, University of Silesia, Poland. At present he works as the head of Computer Systems Department at University of Silesia. Currently his scientific researches focus on computer networks, biometrics and biometric classifier systems, image processing, biomedical imaging and spectral methods of Boolean function analysis. He received Ph. D. and D. Sc. (habilitation) degrees in computer science from AGH University of Science and Technology in Krakow. He has published over 100 scientific papers, which were published in Polish and foreign journals as well as numerous national and international conference presentations and talks. He has also published 2 books and edited 5 books. He was also invited to many program committees of numerous national and international conferences. Professor Porwik has taken part in the capacity of special guest and has been invited as a speaker in conferences and congresses in Poland, Japan and India. Since 2007 he is the Editor-in-Chief of Journal of Medical Informatics & Technologies. He is a reviewer of numerous international journals and fellowships including IEEE Int. J. Circuits, Devices & Systems, IEEE Transactions on Neural Networks, Transactions on Transport Systems Telematics, Int. Journal of Biometrics, American Mathematical Society.