Effective Computer Methods for the Calculation of Rademacher–Walsh Spectrum for Completely and Incompletely Specified Boolean Functions

Bogdan J. Falkowski, *Member, IEEE*, Ingo Schäfer, and Marek A. Perkowski, *Member, IEEE*

***Abstract*—A theory has been developed to calculate the Rademacher–Walsh transform from a cube array specification of incompletely specified Boolean functions. The importance of representing Boolean functions as arrays of disjoint ON- and DC-cubes has been pointed out, and an efficient new algorithm to generate disjoint cubes from nondisjoint ones has been designed. The transform algorithm makes use of the properties of an array of disjoint cubes and allows the determination of the spectral coefficients in an independent way. The programs for both algorithms use advantages of C language to speed up the execution. The comparison of different versions of the algorithm has been carried out. The presented algorithm and its implementation is the fastest and most comprehensive program (having many options) known to us for the calculation of Rademacher–Walsh transform. It successfully overcomes all drawbacks in the calculation of the transform from the design automation system based on spectral methods—the SPECSYS system from Drexel University that uses Fast Walsh Transform.**

***Index Terms*—Algorithms, Rademacher-Walsh transform, Spectral coefficients, Logic design, Cube calculus, Array of disjoint cubes, Sum-of-products expression, Completely and incompletely specified Boolean functions, Standard trivial functions, Orthogonal functions.**

I. Introduction

IN DIGITAL logic design, spectral techniques have been used for more than 30 years. They have been applied to Boolean function classification [9], [22], [23], [36], [37], disjoint decomposition [23], [50]–[52], [54], parallel and serial linear decomposition [10], [22]–[25], [51], [52], [54], spectral translation synthesis (extraction of linear pre- and post-filters) [10], [24]–[26], [28], [51], [52], [54], multiplexer synthesis [23], [31], prime implicant extraction by spectral summation [23], [26], [28], [29], threshold logic synthesis [9], [22], [25], logic complexity [25], [55] and state assignment [25], [53]. Spectral methods for testing of logical networks by verification of the coefficients in the spectrum have been developed [10], [21], [23], [24], [26], [32]–[35], [38], [48], [49]. It should be stressed that an important problem of finding the complement of a Boolean function that has high complexity in the Boolean domain [2], [47] can be solved very easily in the spectral domain because complementing the Boolean function corresponds to changing the sign of every spectral coefficient [22], [23]. Tautology of a Boolean function can be verified by calculating a certain coefficient (DC coefficient). The problem of constructing optimal data compression schemes by spectral techniques has also been considered. The latter approach is very useful for compressing test responses of logical networks and memories [24], [26], [48], [49]. The renewed interest in applications of spectral methods in logic synthesis is caused by their excellent design for testability properties and the possibility of performing the decomposition with gates other than the ones used in most classical approaches. Another area of application is signal processing, especially image processing and pattern analysis [1], [26], [35], [45]. Spectral techniques have also been used for data transmission, especially in the theory of error-correcting codes and for digital filtering [26].

Two design automation systems have used spectral methods as the tool for designing digital circuits [40], [51], [52], [54], [55]. Computation of the spectrum is a complex operation that requires, in the general case, $n2^n$ operations of additions/subtractions when the Fast Walsh Transform [1] is used and the Boolean function has n input variables. In order to store the calculated spectrum, 2^n memory locations are required [1], [23]. The SPECSYS (for SPECtral SYnthesis System) developed at Drexel University on VAX 11/780 uses the Fast Walsh Transform [1] for the calculation of the spectrum and can process Boolean functions having a maximum of 20 input variables [51], [54]. The DIADES design automation system developed at Portland State University [40] does not have any limit on the number of input variables of Boolean functions that can be processed and uses the methods described in this article for the generation of spectral coefficients of Boolean functions.

Logic synthesis for recently developed field-programmable gates arrays (FPGA's), gate arrays, and PLD's creates new requirements for design automation systems because of fundamental architectural differences with re-

Manuscript received April 24, 1990; revised December 31, 1990, and July 11, 1991.

B. J. Falkowski was with Portland State University, Portland, OR. He is now with the Division of Electronic Engineering, Nanyang Technological University, Singapore 2263.

I. Schäfer and M. A. Perkowski are with the Department of Electrical Engineering, Portland State University, Portland, OR 97207.

IEEE Log Number 9107738

0278-0070/92$03.00 © 1992 IEEE

spect to existing technologies. A high demand exists for the methods that produce circuit realizations with EXOR gates [7], [28]. "A four-input XOR (in Xilinx 2000 Logic Cell Arrays) uses the same space and is as fast as a four-input AND gate. · · · Logic design for Xilinx devices is therefore limited by fan-in—not by logic complexity as in PLD's" quoted from [7]. "Any system which flattens functions into 2-level AND-OR form, or which factors based on the "unate paradigm" (as do MIS-II, BOLD, and Synopsys), is going to have problems with strongly non-unate functions like parity, addition, or multiplication. Since these sorts of functions occur frequently in real designs, synthesis tools need reasonable ways of handling them" quoted from [28].

The DIADES design automation system methodology is oriented towards detecting the linear (EXOR) part of a Boolean function. It uses, among others, spectral methods to detect EXOR parts of a Boolean function. Since the DIADES system uses spectral methods together with the programs based on the "unate paradigm," it can easily handle not only functions close to strongly unate but strongly non-unate ones as well. The decomposition of Boolean functions with both pre- and post-linear parts by spectral means leads to highly testable circuit realizations that can be efficiently implemented in several technologies (LHS501 from Signetics, 2000/3000 Logic Cell Arrays from Xilinx, and some EPLD's with EXOR gates). Only spectral methods currently allow for this kind of decomposition [25], [54].

In this paper, the main emphasis is placed on the efficient computer calculation of the Rademacher–Walsh spectrum of Boolean functions since this particular ordering of Walsh transforms is frequently used for logic design [1], [6], [9], [10], [22]–[26]. The ordering of Walsh transforms describes the sequence in which Walsh functions are placed in the transform matrix. There are two Rademacher–Walsh spectra of Boolean functions, and these are known in the literature under the names of R and S spectra [6], [9], [11]–[13], [22]–[26]. In the following material, these spectra are referred to by symbols R and S. The particular coefficients from these spectra are referred to as r_I and s_I, where symbol I is called an index and is used to denote any spectral coefficient from a given spectrum. Both spectra are formed as the product of a $2^n \times 2^n$ Rademacher–Walsh transform matrix T and a 2^n vector representation of a Boolean function F (vector representation of a truth table) [25], [29]. The truth vector for spectrum R is coded by its original values: 0 for false minterms (minterms that have logical values 0), 1 for true minterms (minterms that have logical values 1), and 0.5 for don't care minterms (minterms for which the Boolean function can have an arbitrary logical values 0 or 1). In the case of S spectrum the true minterms are denoted by -1, false minterms by $+1$, and don't care minterms by 0.

This paper shows two *new methods* for the calculation of the Rademacher–Walsh spectrum of incompletely specified Boolean functions. Both these methods can calculate a Walsh spectrum of any ordering since the algorithms are independent of the ordering of the spectral coefficients. Since a direct linear relationship exists between the R and S spectra described in the next section, then this article uses mainly the S spectrum. The *first method*, which allows calculation of the spectrum directly from a Karnaugh map, is introduced here for better understanding of the meaning of spectral coefficients in classical logic terms. The *second method* has been implemented in the DIADES automation system [40].

This paper resolves many important issues concerning the efficient application of spectral methods in computer-aided design of digital circuits. The main obstacle in these applications was, up to now, memory requirements for computer systems. By using the algorithms presented in this article this obstacle has been overcome. Moreover, the methods presented in this article can be regarded as representative of a whole family of methods, and the approach presented can be easily adapted to other transforms used in digital logic design. For example, the adaptation of the *first method* for Adding and Arithmetic Transforms was described in [14], while the adaptation of the *second method* for Reed–Muller Transform was presented in [17]. Both methods are also universal for multiple-valued binary functions and the extension of the *first method* for such functions was presented in [16].

The advantages of the approach presented were possible due to new insight and the formulation of spectral techniques. By investigating the links between spectral techniques and classical logic design methods, this interesting area of research is presented in a simple manner. The real meaning of spectral coefficients in classical logic terms (such as minterms and cubes) is shown. An algorithm is presented to ease the calculation of spectral coefficients for completely and incompletely specified Boolean functions by manipulations directly on Karnaugh maps. All the mathematical relationships between the number of true, false, and don't care minterms and spectral coefficients, as well as between the size of disjoint cubes and spectral coefficients, are stated.

One of the drawbacks of spectral techniques is that practically all the existing algorithms for calculating the spectral coefficients start from a Boolean function, represented either as a list of true minterms (alternatively—a list of false minterms) [22], [23], [25], [26], [51], [52], [54] or as an already minimized sum-of-products Boolean expression (SOPE) [23], [35]. The algorithm presented overcomes this weakness by representing a completely specified Boolean function as a set of disjoint cubes that completely covers this function. A disjoint cube representation of a Boolean function (called "a disjoint cover" and generated from a minimized SOPE) was used to calculate the "autocorrelation" of a Boolean function in [54]. By using a nonunique disjoint-cube representation of a Boolean function, each spectral coefficient can be calculated separately or all the coefficients can be calculated in parallel. These advantages are absent in the existing methods. The possibility of calculating only some coef-

ficients is very important since there are many spectral methods in digital logic design for which the values of only a few selected coefficients are needed. Some examples of such cases in which the entire spectrum need not be computed are: Walsh and Reed–Muller spectral techniques for fault detection [4], [10], [21], [23], [24], [26], [32]–[34], [38], [48], [49]; spectral translation techniques for extracting core functions [22], [23], [29], [30]; designing of multiplexer-based universal-logic modules [31]; prime implicants extraction [26], [29]; estimation of logic complexity [25], [53], [55]; and approximate implementation of logical functions [35].

Most of the current methods in the spectral domain deal only with completely specified Boolean functions. On the other hand, all the algorithms introduced here are valid not only for completely specified Boolean functions but also for functions with don't cares, since don't care minterms can be represented in the form of disjoint cubes as well.

In order to use Boolean functions that are represented as arrays of nondisjoint cubes, a fast algorithm to generate disjoint cubes is presented. The use of the disjoint cube representation of Boolean functions has been found advantageous in many algorithms used in digital logic design [2], [3], [8], [12], [19], [39], [41].

The theory of calculation of the spectral coefficients for incompletely specified Boolean functions is new for both of the methods introduced. The *second method* also allows for the calculation of spectra of a *system of Boolean functions*. When the system of incompletely specified Boolean functions is processed, there is a restriction in [25] that all the functions in the system are assumed to be undefined at exactly the same points (minterms of a Karnaugh map). Optimal completion of don't care minterms for such a system of Boolean functions from the point of view of a minimal number of spectral coefficients different from 0 (and thereby obtaining simpler implementation of such a system of functions) was presented in [25]. However, this is a severe restriction. The *second method* can process not only such functions but any system of completely and incompletely specified Boolean functions. Each function in the system of functions processed by the *second method* can have don't care minterms anywhere in the function domain.

A short description of previously known properties of the classical Rademacher–Walsh transform as applied to Boolean functions should make this article self-sufficient. All properties regarding incompletely specified Boolean functions are new. The formal proofs of these new properties by mathematical induction are trivial and therefore omitted. A number of examples are given that should help to introduce these ideas to people working in the areas of test generation and logic design automation. The use of complicated mathematical formulas, so typical in articles on the subject, is minimized in this presentation. This is important since, unfortunately, up to now the unfamiliarity with the mathematical side of the spectral approach seems to have been too great a hurdle to overcome for finding a fruitful place for practical CAD applications of these ideas.

II. Links between Spectral Techniques and Classical Logic Design

A method of calculating Chow parameters, which are used in the classification of linear logical functions, was shown in [18]. The method was stated only for completely specified Boolean functions. It was based on the computation of the number of agreements minus the number of disagreements between the values in the truth table of a Boolean function in S coding (the minterms of the function are coded according to the S coding) and the values of each successive row of the Rademacher–Walsh matrix T. A minterm in S coding and an element in the row of a transform matrix agree with each other when they have the same value (either 1 or -1); otherwise they disagree. Due to the relationships between the Chow parameters and Rademacher–Walsh spectral coefficients [18], [19], and mutual relationships between both S and R spectra stated for completely specified Boolean functions in [23], the same method can be applied for finding spectral coefficients of only completely specified Boolean functions. However, the agreement/disagreement method, which is suitable for hand calculations, takes into the consideration all 2^n combinations of values of n variables of a Boolean function.

A similar method for hand calculation of the spectral coefficients was proposed in [6]. The advantage of this approach is that only one half of all the combinations of n variables of a Boolean function has to be considered. Thus, the speed of this proposed technique is at last twice the speed of the agreement/disagreement method. However, this method was presented only for completely specified Boolean functions and only for the S spectrum (which was erroneously denoted by symbol R in [6]). Due to mutual relationships between both spectra, this method can be easily extended for the calculation of the R spectrum.

In this paper, all important relationships between spectral coefficients from both spectra and classical logic terms are stated. Moreover, for the first time these relationships are presented not only for completely specified Boolean functions but for incompletely specified Boolean functions as well. The latter problem has been solved by the authors for both S and R spectra. It has also been confirmed that the relationships that bind both these spectra together (see (4) and (5)) are still valid for the spectra calculated for incompletely specified Boolean functions. By expressing spectral coefficients through different formulas (this has never been done thoroughly even for completely specified Boolean functions, and (10)–(13) and (15) are new for such functions) one is able to calculate the spectral coefficients from different available data that makes the methods more flexible.

Let us show more clearly the meaning of r_l and s_l spectral coefficients in classical logic terms. Moreover, let us also expand our considerations for incompletely specified

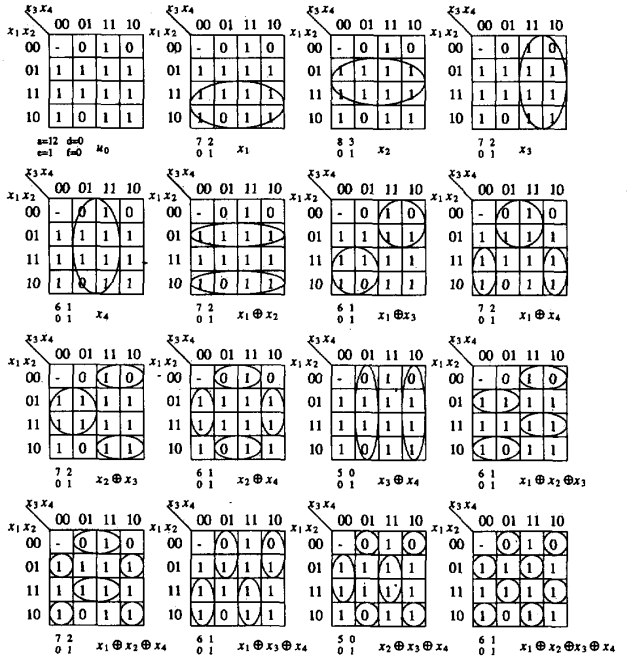


Fig. 1. Standard trivial functions for a four-variable incompletely specified Boolean function.

Boolean functions. A standard trivial function denoted by a symbol u_I describes some area on a Karnaugh map that has an influence on the value of a spectral coefficient s_I. The formal definition of a standard trivial function [22] is given in Property 3.6. All but one standard trivial function for a four variable Boolean function are shown as the circled areas in Fig. 1. The standard trivial function corresponding to the first spectral coefficient s_0 is the whole Karnaugh map and is denoted by symbol u_0. Since for this spectral coefficient the corresponding row in the transform matrix T has all elements equal to 1 (there are no changes of value in the row), this coefficient is called a *direct current coefficient* [23]. The symbol I used in the description of spectral coefficients and standard trivial functions is called an *index*, and the same symbol is used for all the coefficients. The index is composed of subindexes; each spectral coefficient has different subindexes. The *subindexes* of a given spectral coefficient describe the variables of the Boolean function that indicate the area of the standard trivial function corresponding to this spectral coefficient. For example, the standard trivial function described by $x_1 \oplus x_2 \oplus x_3 \oplus x_4$ corresponds to spectral coefficient $s_{1,2,3,4}$ and the general symbol of such a coefficient in s_I.

The following symbols will be used in the equations. Let a_I be the number of true minterms of a Boolean function F in the area for which both the function F and the standard trivial function u_I have the logical value 1; let b_I be the number of false minterms of the Boolean function F in the area for which the function F has the logical value 0 and the standard trivial function u_I has the logical value 1; let c_I be the number of true minterms of the Boolean function F in the area for which the function F has the logical value 1 and the standard trivial function u_I has the logical value 0; let d_I be the number of false minterms of the Boolean function F in the area for which both the function F and the standard trivial function u_I have the logical values 0; let e_I be the number of don't care minterms of the Boolean function F in the area for which the standard trivial function u_I has the logical value 1; and let f_I be the number of don't care minterms of Boolean function F in the area for which the standard trivial function u_I has the logical value 0.

Then, for completely specified Boolean functions having n variables, the following formulas hold for all but the s_0 and r_0 spectral coefficient (when $I \neq 0$) [22]:

$$a_I + b_I + c_I + d_I = 2^n \tag{1}$$

and [22]

$$a_I + b_I = c_I + d_I = 2^{n-1}. \tag{2}$$

For the s_0 and r_0 spectral coefficients [22]:

$$a_0 + b_0 = 2^n \tag{3}$$

as c_0 and d_0 are both zero.

The relationships between spectra R and S are as follows [23]:

$$r_0 = \tfrac{1}{2}(2^n - s_0) \tag{4}$$

and [23]

$$r_I = -\tfrac{1}{2} s_I \tag{5}$$

when $I \neq 0$.

Accordingly, for incompletely specified Boolean functions having n variables, the following formulas hold for all but the s_0 and r_0 spectral coefficient (when $I \neq 0$):

$$a_I + b_I + c_I + d_I + e_I + f_I = 2^n \tag{6}$$

and

$$a_I + b_I + e_I = c_I + d_I + f_I = 2^{n-1} \tag{7}$$

For the s_0 and r_0 spectral coefficients:

$$a_0 + b_0 + e_0 = 2^n \tag{8}$$

as c_0, d_0, and f_0 are all zero.

Subsequently, (1)–(8) are used, where necessary, in order to derive different alternative expressions for spectral coefficients and the final formulas are presented.

The s_0 spectral coefficient for completely specified Boolean functions can be defined in the following alternative ways [22]:

$$s_0 = 2^n - 2a_0 \tag{9}$$

or

$$s_0 = 2b_0 - 2^n \tag{10}$$

or [22]

$$s_0 = b_0 - a_0. \tag{11}$$

The s_I spectral coefficients (when $I \neq 0$) for completely specified Boolean functions can be calculated according

to any of the following formulas:

$$s_I = 2(a_I + d_I) - 2^n \tag{12}$$

or

$$s_I = 2^n - 2(b_I + c_I) \tag{13}$$

or [22]

$$s_I = 2(a_I - c_I) \tag{14}$$

or

$$s_I = 2(d_I - b_I) \tag{15}$$

or [22]

$$s_I = (a_I + d_I) - (b_I + c_I). \tag{16}$$

Similarly, for completely specified Boolean functions having n variables, the r_0 spectral coefficient can be defined in three ways from (9) to (11) by using (4). The r_I spectral coefficients (when $I \neq 0$) for completely specified Boolean functions having n variables can be defined in five ways from (12) to (16) by using the relationship of (5).

Let us now expand our considerations for incompletely specified Boolean functions having n variables. Then, the s_0 spectral coefficient can be defined in the following alternative ways:

$$s_0 = 2^n - 2a_0 - e_0 \tag{17}$$

or

$$s_0 = 2b_0 + e_0 - 2^n \tag{18}$$

or

$$s_0 = b_0 - a_0 \tag{19}$$

The s_I spectral coefficients (when $I \neq 0$) for incompletely specified Boolean functions can be calculated according to the following formulas:

$$s_I = 2(a_I + d_I) + e_I + f_I - 2^n \tag{20}$$

or

$$s_I = 2^n - 2(b_I + c_I) - (e_I + f_I) \tag{21}$$

or

$$s_I = 2(a_I - c_I) + e_I - f_I \tag{22}$$

or

$$s_I = 2(d_I - b_I) + f_I - e_I \tag{23}$$

or

$$s_I = (a_I + d_I) - (b_I + c_I). \tag{24}$$

Similarly, for incompletely specified Boolean functions having n variables, the r_0 spectral coefficient can be defined in three ways from (17) to (19) by using (4). The r_I spectral coefficients (when $I \neq 0$) for completely specified Boolean functions having n variables can be defined in five ways from (20) to (24) by using the relationship of (5).

At this point, it is interesting to note the following properties of the above equations. First, (4) and (5) are valid for spectra calculated for both completely and incompletely specified Boolean functions. Secondly, (11) and (19) for the calculation of the direct current (s_0) and the corresponding formulas for the calculation of the r_0 coefficient are exactly the same for both completely and incompletely specified Boolean functions. The same phenomenon occurs in (16) and (24) and in the corresponding formulas for the calculation of r_I (when $I \neq 0$) spectral coefficients. A number of don't care minterms e_I and f_I are eliminated from these formulas. Of course, this does not mean that the spectral coefficients for an incompletely specified Boolean function do not depend on the number of don't care minterms. They do, but this dependence is included in (6)–(8).

Example 1: As a numerical example, consider a four-variable incompletely specified Boolean function for which all standard trivial functions and values of all corresponding a_I, d_I, and f_I are shown in Fig. 1. According to (17) and (20), the s_I spectral coefficients for this function are as follows:

$$s_0 = 16 - 24 - 1 = -9, \quad s_1 = 18 + 1 - 16 = 3$$

$$s_2 = 22 + 1 - 16 = 7, \quad s_3 = 18 + 1 - 16 = 3$$

$$s_4 = 14 + 1 - 16 = -1, \quad s_{12} = 18 + 1 - 16 = 3$$

$$s_{13} = 14 + 1 - 16 = -1, \quad s_{14} = 18 + 1 - 16 = 3$$

$$s_{23} = 18 + 1 - 16 = 3, \quad s_{24} = 14 + 1 - 16 = -1$$

$$s_{34} = 10 + 1 - 16 = -5,$$

$$s_{123} = 14 + 1 - 16 = -1, \quad s_{124} = 18 + 1 - 16 = 3$$

$$s_{134} = 14 + 1 - 16 = -1, \quad s_{234} = 10 + 1 - 16 = -5,$$

$$s_{1234} = 14 + 1 - 16 = -1.$$

As can be seen from the above example, the *first method* can generate spectral coefficients in any ordering of Walsh functions or only some subset of them.

The relations that have been determined in this section allow the derivation of many properties of spectral coefficients that are important in their applications to the analysis and synthesis of logical circuits. The next section describes the classical approach to the calculation of spectra of Boolean functions. For illustration, instead of Fast Walsh Transform, the matrix multiplication method is presented. The advantages of introduced algorithm over Fast Transforms are shown. All essential terms and properties of the Rademacher–Walsh S and R spectra are included.

III. Basic Properties of Rademacher–Walsh Spectrum

The Rademacher–Walsh spectra S and R of an n-variable Boolean function are alternative canonical representations of the Boolean function. Other related canonical descrip-

tions of Boolean functions are the Adding and Arithmetic forms [14] and the Reed–Muller from [4], [17], [20]. The latter form, however, is a canonical representation of only completely specified Boolean functions.

The Rademacher–Walsh spectrum S is formed as the product of a $2^n \times 2^n$ Rademacher–Walsh matrix T and a 2^n vector representation M of a Boolean function F. When the Boolean function is represented as a truth table of all minterms, M will be either the $\langle +1, -1 \rangle$ vector representation of the truth table for a completely specified Boolean function, or the $\langle +1, 0, -1 \rangle$ vector representation of the truth table for an incompletely specified Boolean function [1], [22], [23], [25], [26], [34]. In the coding scheme the conventional $\langle 0, 1, - \rangle$ values (false, true and don't care minterms) correspond to $\langle +1, -1, 0 \rangle$ codings, respectively ("$-$" stands for "don't care"). This type of coding of the truth vector is called the S coding, and the coded truth vector is denoted by M. The values of the minterms in the original truth vector and the coded vector M are ordered according to the straight binary code (SBC) of variables describing these minterms. For example, the first entry in the vector is the logical value of the minterm that is described by all negated variables of a Boolean function (minterm 0), the second entry in the truth vector is the logical value of the second minterm that is described by all negated variables except x_1 (minterm 1, the least significant bit of SBC = 1), etc. The Rademacher–Walsh matrix T represents the Walsh functions in Rademacher ordering. The rules for recursive generation of such matrices are described in [25].

The Rademacher–Walsh spectrum R results from another coding of the truth table of a Boolean function in which the conventional $\langle 0, 1, - \rangle$ values correspond to $\langle 0, 1, 0.5 \rangle$. This Rademacher–Walsh spectrum is called the R spectrum, and this type of coding is called the R coding. The relationships between the spectral coefficients of the S and R spectra were given in the previous section.

Besides the matrix method presented here, recursive algorithms, data flow-graph methods, and parallel calculations similar to the Fast Fourier Transform have also been used to calculate the Rademacher–Walsh and other related transforms [1], [23], [26], [54]. All the methods mentioned reduce the necessary number of calculations from $n \times n$ to $n \times \log_2 n$, but still require an excessive computer memory. They also have some undesirable properties, discussed mainly in Section X, that are overcome by using the second spectrum calculation method introduced here.

The *principal properties* of the coefficients of the S spectrum for completely specified Boolean functions for the four well-known Walsh-type transform matrices are shown below according to [1], [9], [22], [23], [25], [26]. The Properties 3.7, 3.9, 3.10, 3.13, and 3.17 are new.

3.1) The transform matrix of each ordering of the Walsh functions is *complete* and *orthogonal;* therefore, there is no information lost in the S and R spectra concerning the minterms of the Boolean function F.

3.2) Only the Hadamard–Walsh matrix describing the Hadamard–Walsh transform has the *recursive Kronecker product structure*. Other possible variants of the Walsh transforms, described by the corresponding matrices, are known in the literature as the Walsh–Kaczmarz, Rademacher–Walsh, and Walsh–Paley transforms.

3.3) Only the Rademacher–Walsh matrix is not *symmetric;* all other variants of Walsh matrices are symmetric, so that, disregarding a scaling factor, the same matrix can be used for both the forward and inverse transform operations.

3.4) Each spectral coefficient s_I or r_I is described by its *order, subindexes* and *magnitude*. The order of the spectral coefficient is equal to the number of subindexes, and the subindexes are the subscripts of all variables of a standard trivial function corresponding to the coefficient. The magnitude of a spectral coefficient is its value. In the sequel, i, j, and k denote subindexes, and the order is denoted by o.

3.5) When the classical matrix multiplication method is used to generate the spectral coefficients for different Walsh transforms (different T matrices represent different Walsh functions with different orderings), the only difference in the result is the *ordering* of the spectral coefficients. The coded vector M corresponding to the original truth vector of a Boolean function is the same for all orderings of Walsh functions. The values of the coefficients s_I and r_I with the same subindices are the same for every ordering of Walsh transforms.

3.6) Each spectral coefficient (either s_I or r_I) gives a *correlation value* between the Boolean function F and a *standard trivial function* u_I corresponding to the coefficient. The *standard trivial functions* for the spectral coefficients are, respectively: for the dc coefficients (direct current coefficients) s_0 or r_0, the universe of the Boolean function F denoted by u_0; for the first order coefficients s_I or $r_I (I = i, i \neq 0)$, the variable x_i of the Boolean function F denoted by u_i; for the second order coefficients s_I or $r_I (I = ij, i \neq 0, j \neq 0, i \neq j)$, the Exclusive-OR function between variables x_i and x_j of the Boolean function F denoted by u_{ij}; for the third-order coefficients s_I or $r_I (I = ijk, i \neq 0, j \neq 0, k \neq 0, i \neq j, i \neq k, j \neq k)$, the Exclusive-OR function between variables x_i, x_j, and x_k of the Boolean function F denoted by u_{ijk}, etc.

3.7) The number of spectral coefficients of zth order is equal to $C_n^z = \binom{n}{z}$, where n is the number of variables of a Boolean function.

3.8) For a completely specified Boolean function the maximal value of any individual spectral coeffi-

cient s_l is $+2^n$ and the minimal value is -2^n. These values occur when the Boolean function F is equal to either a standard trivial function u_l (sign +) for the maximal value or to its complement (sign −) for the minimal one. In either case, all the remaining spectral coefficients have zero values because of the orthogonality of the transform matrix T.

3.9) The maximal value of any spectral coefficient r_l except r_0 is $+2^{n-1}$ and will result when the Boolean function F is equal to complement of a standard trivial function u_l. The minimal value is -2^{n-1} and will result when F is equal to a standard trivial function u_l. In either case, all the remaining spectral coefficients have zero values because of orthogonality of the transform matrix T.

3.10) For an incompletely specified Boolean function the maximal value of any individual spectral coefficient s_l is $+2^n - 1$ and the minimal one is $-2^n + 1$. These occur when the Boolean function F is equal to either a standard trivial function for the maximal value of s_l or to its complement for the minimal value, in all but one minterm.

3.11) The maximal value of the r_0 spectral coefficient is $+2^n$ and will result when the Boolean function F is a tautology. The minimal value is -2^n and will result when F is equal to the complement of the tautology. The tautology is the logical function for which all the minterms are true.

3.12) When, for more than half of the spectral coefficients of any completely specified Boolean function F, the majority of the minterms have the same logical values as the minterms of standard trivial functions, the sum of all of the coefficients of the S spectrum has a maximum value equal to $+2^n$. When, for more than half of the spectral coefficients, the majority of minterms of F have the complemented logical values to the minterms of standard trivial functions, the sum of all of the coefficients of the S spectrum has a minimal value equal to -2^n.

3.13) For any incompletely specified Boolean function the sum of all the coefficients of the S spectrum has a maximal value of $+2^n - 1$ and a minimal value of $-2^n + 1$. The maximal or minimal value happens when the Boolean function has exactly one don't care minterm and all the spectral coefficients follow the rule found in Property 3.12.

3.14) With the exception of u_0, every standard trivial function u_l corresponding to an n variable Boolean function F has the same number of true and false minterms. That number is equal to 2^{n-1}.

3.15) For each true minterm the coefficients from the spectrum S are $s_0 = 2^n - 2$, and all remaining $2^n - 1$ spectral coefficients s_l are equal to ± 2.

T																M		S	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0		-9	s_0
1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1		-1	s_4
1	1	1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	-1		3	s_3
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	-1		7	s_2
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1		3	s_1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1		-5	s_{34}
1	1	-1	-1	1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	-1		-1	s_{24}
1	1	-1	-1	-1	-1	1	1	1	1	-1	-1	-1	-1	1	1	-1	=	3	s_{23}
1	-1	1	-1	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	1		3	s_{14}
1	-1	1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1		-1	s_{13}
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	-1		3	s_{12}
1	1	-1	-1	-1	-1	1	1	-1	-1	1	1	1	1	-1	-1	-1		-5	s_{234}
1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	-1	1	-1	-1		-1	s_{134}
1	-1	-1	1	1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	-1		3	s_{124}
1	-1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	1	-1	-1		-1	s_{123}
1	-1	-1	1	-1	1	1	-1	-1	1	1	-1	1	-1	-1	1	-1		-1	s_{1234}

Fig. 2. Calculation of Rademacher-Walsh S spectrum.

The method for choosing the signs of spectral coefficients is defined in Section V (Properties 5.6–5.8).

3.16) The spectrum S of each false minterm is given by $s_l = 0$.

3.17) For each don't care minterm the coefficients from the spectrum S are $s_0 = 2^{n-1} - 1$, and all remaining $2^n - 1$ spectral coefficients s_l are equal to ± 1. The method for choosing the signs of spectral coefficients is defined in Section V (Properties 5.6–5.8).

Example 2: An example of the calculation of Rademacher–Walsh spectrum S of a four-variable incompletely specified Boolean function is shown in Fig. 2. The matrix T describes the discrete Walsh functions in Rademacher ordering. The coding truth vector M represents the values of minterms of a Boolean function in the S coding. In this example, the function from Fig. 1 is used. One can notice easily that the first entry in the vector M has value 0 since it corresponds to the logical value don't care of the minterm described by all variables negated on Karnaugh maps from Fig. 1. Since the Boolean function has only one don't care minterm, all other entries in the vector M are either $+1$ or -1. The obtained spectrum is exactly the same as the one obtained by the *first method*.

The second new method of calculating spectral coefficients of Boolean functions described in this article needs the representation of Boolean functions in the form of arrays of disjoint cubes. The method gives the correct values of spectral coefficients independently from the shape and size of disjoint cubes in the array of cubes as long as all the minterms of the function are covered only once by a cube. The next section describes the algorithm that generates this kind of representation of Boolean functions.

IV. Algorithm to Generate Disjoint Cubes

The calculation time for the Rademacher–Walsh spectrum increases linearly with the number of disjoint cubes. Thus, the determination of a quasi-minimal number of

TABLE I
COMPARISON OF THE NUMBER OF DISJOINT CUBES FOR THE MCNC BENCHMARK FUNCTIONS

	Input Variables	Output Variables	ESPRESSO -Ddisjoint	Our Program
b12	15	9	654	57
clip	9	5	185	162
inc	7	9	34	34
misex1	8	7	32	15
misex2	25	18	29	28
rd53	5	3	32	31
rd73	7	3	141	127
sao2	10	4	157	98
5xp1	7	10	106	70
9sym	9	1	189	166

cubes in the disjoint cube representation of a Boolean function is crucial for the effective calculation of its spectrum.

Several algorithms for the generation of arrays of disjoint ON- and DC-cubes (if any) or an array of disjoint OFF-cubes were described and used in PALMINI [39], UMINI [3], EXORCISM [19], EXORCISM-MV [41], and ESPRESSO [2]. Here the algorithm and its implementation as described in [17] and [18] are improved. The random ordering of the cubes is replaced by an array of cubes in which the cubes are sorted according to their size. The sorting can be performed in a short computation time by using a skip list [43] to determine the relative sizes of cubes. Thus, the new algorithm (Algorithm 1) compares the largest cube with all others, starting from the smallest one. In the next step, the second largest cube is taken and compared to all smaller ones, etc. As a last step of the algorithm, the cubes are merged, where possible, to obtain a smaller total number of disjoint cubes. When the new algorithm is applied to a system of Boolean functions, it always tries to minimize the total number of disjoint cubes describing the functions in the system.

Example 3: The new algorithm is applied to the function for Example 1 and [18]. The steps of the execution are illustrated in Fig. 3.

A '*' in the array of cubes indicates that the disjoint sharp operation ($\#_j$) [7], [46] has to be performed between those two cubes. If the two chosen cubes in Fig. 3(d) are in a different order, the result cannot be merged to a smaller number of cubes. In order to find the optimal solution in every case, a branching for each sharp operation would be necessary, but is not implemented in our algorithm.

In Table I the new algorithm is compared to the option *-Ddisjoint* of ESPRESSO [2]. The functions shown in Table I have been taken from the MCNC benchmarks. The functions have been minimized by ESPRESSO before calculating their disjoint representation. The execution time for our algorithm was always less than one second, while ESPRESSO took up to 300 s (b12).

The second and third columns of Table I give the number of input/output variables of the MCNC benchmark functions, listed in the first column. The fourth column gives the number of disjoint cubes obtained by ESPRESSO using the *-Ddisjoint* option. The right column shows the number of disjoint cubes obtained by our algorithm. The number of disjoint cubes obtained by our program is usually better than the ones obtained by ESPRESSO.

The next two sections describe the properties used to develop the computer method for generating the Rademacher–Walsh spectra for completely and incompletely specified Boolean functions (Section V) and for systems of completely and incompletely specified Boolean functions (Section VI).

V. AN ARRAY METHOD FOR THE CALCULATION OF SPECTRUM OF A BOOLEAN FUNCTION

An algorithm already exists for calculating spectral coefficients for completely specified Boolean functions directly from a sum-of-products Boolean expression [23], [35]. When the implicants are not mutually disjoint, this algorithm requires an additional correction to calculate the exact values of spectral coefficients for minterms of Boolean function F that are included more than once in some implicants. By using a representation of a Boolean function in the form of an array of disjoint cubes one can apply the existing algorithm without having to perform correction operations because for an array of disjoint cubes as input data the exact values of spectral coefficients can be calculated immediately. Here the extension of the algorithm to incompletely specified Boolean function is proposed.

In what follows the properties of the existing algorithm are rewritten in notation corresponding to our representation of Boolean functions with n variables in the form of arrays of disjoint cubes. This is the first time all the properties describing incompletely specified Boolean functions have been presented.

Definition 1: The *cube of degree m* is a cube that has m literals that can be either in affirmation or negation (i.e., m is equal to the sum of the number of zeros and ones in the description of a cube). Let symbol p denote the number of X's in the cube and let n denote the number of variables of a Boolean function. Then, $n = m + p$.

Example 4: Consider the cube $1X\,00$. It is a cube of degree 3 since three of the literals describing this cube are either in affirmation (x_1) or negation (x_3 and x_4). The cube does not depend on literal x_2.

Definition 2: The *partial spectral coefficient* of an ON- or DC-cube with degree m of a Boolean function F is equal to the value of the spectral coefficient that corresponds to the contribution of this cube to the full n-space spectrum of the Boolean function F. 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.

Example 5: Consider Table II as representing the array method of calculating spectral coefficients. The considered array of disjoint cubes is the result of Example 3.

TABLE II

$x_1x_2x_3x_4$	S_0	S_1	S_2	S_3	S_4	S_{12}	S_{13}	S_{14}	S_{23}	S_{24}	S_{34}	S_{123}	S_{124}	S_{134}	S_{234}	S_{1234}
$X1XX$ ON	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0
$X011$ ON	12	0	−4	4	4	0	0	0	4	4	−4	0	0	0	−4	0
$10X0$ ON	12	4	−4	0	−4	4	0	4	0	−4	0	0	4	0	0	0
0000 DC	7	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
	−9	3	7	3	−1	3	−1	3	3	−1	−5	−1	3	−1	−5	−1

Each row in this table shows the partial spectral coefficients of either ON- or DC-cubes of a Boolean function. The function in the example has four partial spectra, which is equal to the number of disjoint ON- and DC-cubes describing this function (*npsc* = 4).

Suppose arrays of disjoint ON- and DC-cubes that fully define Boolean function F are given. Then each cube of degree m can be treated as a minterm within its particular reduced m-space of function F. Let us recall that the spectrum of each true minterm is given by $s_0 = 2^n - 2$, and all remaining $2^n - 1$ coefficients are equal to ± 2 (Property 3.15). Similarly, the spectrum of each don't care minterm is given by $s_{DC0} = 2^n - 1$, and all the remaining $2^{n-1} - 1$ coefficients are equal to ± 1 (Property 3.17). The symbols s_{DCI} denote spectral coefficients corresponding to DC-cubes (when $I = 0$, the symbol S_{DC0} denotes a direct current spectral coefficient corresponding to a DC-cube).

Cubes of degree m have the following properties.

5.1) The contribution of an ON-cube of degree m to the full n-space spectrum of function F (where n is the number of variables in the function F) is related as follows:

$$s_0 \text{ in full } n\text{-space} = 2^n - 2 \times 2^p \tag{25}$$

and

$$s_I \text{ in full } n\text{-space} = s_I \text{ in } m\text{-space} \times 2^p \tag{26}$$

where $I \neq 0$.

5.2) The contribution of a DC-cube of degree m to the full n-space spectrum of function F is related as follows:

$$s_{DC0} \text{ in full } n\text{-space} = 2^{n-1} - 2^p \tag{27}$$

and

$$s_{DCI} \text{ in full } n\text{-space} = s_{DCI} \text{ in } m\text{-space} \times 2^p$$

where $I \neq 0$. (28)

where $I \neq 0$.

Notice that when the above formulas are applied to minterms (i.e., for $m = n$, and $p = 0$) they reduce to Properties 3.15 and 3.17. The contribution of a DC-cube of degree m is equal to one half of the contribution of an ON-cube that has the same degree m. Moreover, the contribution of ON- or DC-cubes of degree m to the full n-space spectrum of function F can be expressed for s_0 as the absolute value of the sum of all negative spectral coefficients corresponding to these cubes.

Equations (26) and (28) determine the absolute values of those partial spectral coefficients s_I that are not equal to zero for a given cube. Properties 5.3–5.5 determine the signs of the partial spectral coefficients, and whether some of them are equal to zero.

Example 6: Consider Table II again. The value of partial spectral coefficient s_0 corresponding to the ON-cube $10X0$ ($n = 4$, $p = 1$) is equal to $2^4 - 2 \times 2^1 = 12$ according to (25). The absolute values of those partial spectral coefficients s_I that are not equal to zero are calculated according to (26) and are equal to $2 \times 2^1 = 4$.

The value of partial spectral coefficient s_0 corresponding to the DC-cube 0000 ($n = 4$, $p = 0$) is equal to $2^3 - 2^0 = 7$ according to (27). The absolute values of those partial spectral coefficients s_I that are not equal to zero are calculated according to (28) and are equal to $1 \times 2^0 = 1$.

The following properties determine which partial spectral coefficients have values zero for an ON- or DC-cube of the degree m.

5.3) If in a given cube the x_i variable of a Boolean function is denoted by the symbol "X," then all of the partial spectral coefficients s_I whose indexes I contain the subindex i are equal to 0.

5.4) If in a given cube each of the variables of a Boolean function x_i, x_j, x_k, etc., from the complete set of all variables of the function is denoted by symbol "X," then every partial spectral coefficient s_I whose index I contains the subindexes i, j, k, etc., is equal to 0.

5.5) For an ON- or DC-cube of the degree m the number of nonzero partial spectral coefficients is equal to 2^{n-p}, except for $p = n - 1$ when there is only one nonzero partial spectral coefficient.

Example 7: Consider Table II again. The variable x_3 is denoted by symbol X in the cube $10X0$. Then, by Property 5.3, the values of all partial spectral coefficients with subindex 3 are equal to zero. Therefore, $s_3 = s_{13} = s_{23} = s_{34} = s_{123} = s_{134} = s_{234} = s_{1234} = 0$. For this cube, by Property 5.5, the number of partial spectral coefficients different from zero is equal to $2^{4-1} = 8$.

The cube $X1XX$ has three variables denoted by the X symbols: x_1, x_3, and x_4. Therefore, by Property 5.4 and Property 5.5, only the partial spectral coefficient s_2 is different from zero.

The following properties describe the signs of each partial spectral coefficient s_I, where $I \neq 0$, and are valid for ON- and DC-cubes of any degree:

5.6) If in a given cube the x_i variable of a Boolean function is in affirmation, then the sign of the corresponding first-order coefficient is positive; otherwise for a variable that is in negation, the sign of the corresponding first-order coefficient is negative. If in a given cube the x_i variable of a Boolean function is in affirmation, then the sign of the corresponding first-order coefficient is positive; otherwise for a variable that is in negation, the sign of the corresponding first-order coefficient is negative.

5.7) The signs of all even-order coefficients are given by multiplying the signs of the related first-order coefficients by -1.

5.8) The signs of all odd-order coefficients are given by multiplying the signs of the related first-order coefficients.

Example 8: Consider Table II again. In the ON-cube $10X\ 0$ the variable x_1 is in affirmation, while the variables x_2 and x_4 are in negation. Therefore, by Property 5.4 the sign of the partial spectral coefficient s_1 is positive and the signs of partial spectral coefficients s_2 and s_4 are negative.

The signs of second-order coefficients are determined by Property 5.7. The sign of the even-order partial spectral coefficient s_{12} of cube $10X\ 0$ is positive, since the sign is determined by the product of the related first-order coefficients, s_1 and s_2, times -1, i.e., $(-1) \times 1 \times (-1) = 1$.

The signs of the third-order coefficients are determined by Property 5.8. The signs of the partial spectral coefficient s_{124} of the same cube is positive since it is determined according to Property 5.8 as the product of the related first-order coefficients, s_1, s_2, and s_4 and the result is $1 \times (-1) \times (-1) = 1$.

The algorithm is as follows:

Algorithm 2: Calculation of spectral coefficients for completely and incompletely specified Boolean functions.

1. For each ON- and DC-cube of degree m, calculate the value and sign of the contribution of this cube to the full n-space spectrum according to the properties described previously.
   The values of all spectral coefficients s_I, except s_0, are equal to the sum of all of the contributions to the spectral coefficients from all ON- and DC-disjoint cubes from an array of cubes.
3. For a completely specified Boolean function the value of the dc spectral coefficient s_0 is equal to the sum of all of the partial spectral coefficients corresponding to all of the disjoint ON-cubes describing the function, plus the correction factor $-(k-1) \times 2^n$, when k is the number of disjoint cubes in the array of ON-cubes.
4. For an incompletely specified Boolean function the value of the dc spectral coefficient s_0 is equal to the sum of all of the partial spectral coefficients corresponding to all of the disjoint DC-cubes describing the function, plus the correction factor $-(w-1) \times 2^n$, where w is the number of disjoint cubes in the array of DC-cubes.
5. For an incompletely specified Boolean function the value of the DC spectral coefficient s_0 is equal to the sum of all of the partial spectral coefficients corresponding to all of the disjoint ON- and DC-disjoint cubes describing the function, plus the correction factor $-(k-1) \times 2^n - w \times 2^{n-1}$, where k is the number of disjoint ON-cubes, and w is the number of disjoint DC-cubes.

The correction factor $-(k-1) \times 2^n$ compensates for the fact that the cubes over the complete n-space have been added k times during the calculation of the k partial spectral coefficients. A similar explanation applies to DC-cubes as well.

Of course, the algorithm can calculate each coefficient separately or in parallel. If some of the 2^n spectral coefficients are not needed for a particular application, then a reduced number of operations can be performed.

Example 9: An example of the calculation of the S spectrum for the four-variable incompletely specified Boolean function is shown in Table II. The function in this example is the same as the one used in Examples 1, 2, and 3. Fig. 3 showed the stages of the execution of the algorithm generating disjoint cube representation for the same function. Fig. 3(f) showed the input data for the algorithm for this section. The array of disjoint cubes representing the function is repeated from Fig. 3(f) as the first column in Table II. The values and signs of all the partial spectral coefficients for this function are determined by Properties 5.1–5.8. The results of the application of these properties are shown for two cubes.

The spectral coefficients of the first ON-cube in Table II (cube $10X\ 0$ of degree $m = 3$) are as follows:

1) within its own m-space, treated as a single minterm;

$$s_0 = 6,\quad s_1 = 2,\quad s_2 = -2,\quad s_3 = 0$$
$$s_4 = -2,\quad s_{12} = 2,\quad s_{13} = 0,\quad s_{14} = 2$$
$$s_{23} = 0,\quad s_{24} = -2,\quad s_{34} = 0,\quad s_{123} = 0$$
$$s_{124} = 2,\quad s_{134} = 0,\quad s_{234} = 0,\quad s_{1234} = 0.$$

2) within the full n-space of Boolean function F (partial spectral coefficients);

$$s_0 = 12,\quad s_1 = 4,\quad s_2 = -4,\quad s_4 = -4$$
$$s_{12} = 4,\quad s_{14} = 4,\quad s_{24} = -4,\quad s_{124} = 4.$$

On the other hand, the spectral coefficients of the DC-cube in Table II (cube 0000 of degree $m = 4$, i.e., single minterm) are as follows:

1) within its own m-space, treated as a single minterm;

$$s_0 = 7,\quad s_1 = -1,\quad s_2 = -1,\quad s_3 = -1$$
$$s_4 = -1,\quad s_{12} = -1,\quad s_{13} = -1,\quad s_{14} = -1$$

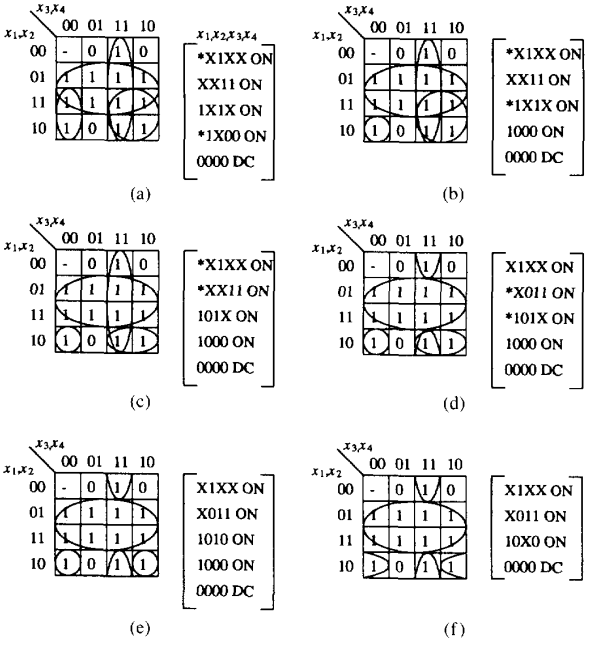


Fig. 3. Stages of the execution of the algorithm to generate a disjoint cube representation. (a) Input array. (b) Cube [4] $\#_j$ cube [1]. (c) Cube [2] $\#_j$ cube [1]. (d) Cube [3] $\#_j$ cube [1]. (e) Cube [3] $\#_j$ cube [2]. (f) Merge.

$$s_{23} = -1, \quad s_{24} = -1, \quad s_{34} = -1, \quad s_{123} = -1,$$
$$s_{124} = -1, \quad s_{134} = -1, \quad s_{234} = -1,$$
$$s_{1234} = -1$$

2) within the full n-space of Boolean function F (partial spectral coefficients): the same as within its own m-space since it is a single minterm.

In order to obtain the values of all of the spectral coefficients of the whole function, except s_0, the columns of partial spectral coefficients corresponding to all cubes describing the function are added (step 2 of the algorithm). The value of s_0 is obtained by the addition of all partial spectral coefficients with the correction factor (step 5 of the algorithm). Since the considered function is incompletely specified and not all the minterms are don't cares, the steps 3 and 4 are not performed.

The resulting spectrum is shown in the bottom row of Table II and, as can be easily checked, is exactly the same as the one obtained by using the standard trivial functions in Example 1 and by matrix multiplication in Example 2.

VI. Spectral Coefficients Calculation for Systems of Boolean Functions

The algorithms from Sections II and V can be modified easily to calculate Walsh spectra of systems of Boolean functions. Because the method from Section II (Karnaugh maps) is limited to six variables, only the extension of the Algorithm 2 is presented.

The calculation of a Walsh spectrum for a system of completely specified Boolean functions was presented in [25] for the R coding. There, the calculation of the R spectrum of a system of incompletely specified Boolean functions is considered, with the following restriction.

Restriction 1: When a system of incompletely specified Boolean functions has don't care minterms, then all of the functions of the system have the same don't care minterms (i.e., the same cells of Karnaugh maps are not specified in every function of the system).

The method presented in [25], however, has all the drawbacks of the classical approach of spectral methods since it uses matrix calculation methods.

In this section the representation of systems of Boolean functions with the above restriction on a system of incompletely specified Boolean functions is presented for the first time for S coding. Since the method with the restriction is of little practical use, the representation of systems of incompletely specified Boolean functions that can have any don't care minterms is introduced. When applied to a system of Boolean functions, the method still has all the advantages described in the previous section.

Let us assume that the functions in the system are in the order: $F[1], F[2], F[3], \cdots, F[b]$, where b is the number of the functions in the system and the function $F[b]$ is on the rightmost position in the system. Then, for the system of completely specified Boolean functions and for the system of incompletely specified Boolean functions, with Restriction 1, the following properties hold:

6.1) The contribution of the spectrum of the function $F[i]$, $i = 1, 2, \cdots, b$ to the total spectrum of a system of Boolean functions S_{TOT} is equal to the spectrum $S_{[i]}$ of the function $F[i]$ calculated by Algorithm 2, which in turn has to be modified by (29).

6.2) The total spectrum of a system of Boolean functions S_{TOT} is equal to the sum of all the modified spectra of all the Boolean functions in the system.

The contribution of the spectrum of the "ith" function $F[i]$ to the total spectrum of a system of b Boolean functions is denoted in (29) by $S^*_{I[i]}$, and the spectrum of the "ith" function calculated by Algorithm 1 is denoted by $S_{I[i]}$.

$$S^*_{I[i]} = 2^{b-i} \times S_{I[i]}. \tag{29}$$

When the more general and practical case of a system of incompletely specified Boolean functions having arbitrary don't care minterms is considered, the system has to be represented by two spectra—one corresponding to the don't care minterms of the system, the second corresponding to the true minterms. The need for two separate spectra for a system of arbitrary incompletely specified Boolean functions arises from the properties of the Rademacher–Walsh matrix T. If Properties 6.1 and 6.2 were applied to don't care and true minterms of an arbitrary system of incompletely specified Boolean functions,

TABLE III

	S_0	S_1	S_2	S_3	S_4	S_{12}	S_{13}	S_{14}	S_{23}	S_{24}	S_{34}	S_{123}	S_{124}	S_{134}	S_{234}	S_{1234}
$S_{[2]}$	−10	2	6	2	−2	2	−2	2	2	−2	−6	−2	2	−2	−6	−2
$S_{[1]}$	6	2	−2	−2	2	2	−6	6	−2	2	10	2	−2	−2	2	−2
$S^*_{[1]}$	12	4	−4	−4	4	4	−12	12	−4	4	20	4	−4	−4	4	−4
$S_{TOT\,ON}$	2	6	2	−2	2	6	−14	14	−2	2	14	2	−2	−6	−2	−6

TABLE IV

	S_0	S_1	S_2	S_3	S_4	S_{12}	S_{13}	S_{14}	S_{23}	S_{24}	S_{34}	S_{123}	S_{124}	S_{134}	S_{234}	S_{1234}
$S_{[2]}$	7	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
$S_{[1]}$	6	0	2	2	2	0	0	0	−2	−2	−2	0	0	0	2	0
$S^*_{[1]}$	12	0	4	4	4	0	0	0	−4	−4	−4	0	0	0	4	0
$S_{TOT\,DC}$	19	−1	3	3	3	−1	−1	−1	−5	−5	−5	−1	−1	−1	3	−1

then the original system of functions would not be retrieved when the inverse transform is applied. For example, the contribution of the don't care minterm for the function $F[b-1]$ to the total spectrum of the system of Boolean functions would be, in a case of using both Properties 6.1 and 6.2, the same as the contribution of the true minterm of the function $F[b]$. Therefore, Properties 6.1 and 6.2 can be applied to an arbitrary system of incompletely specified Boolean functions only after representing each of the functions in this system by two arrays of cubes: one containing only don't cares minterms, the other containing only true minterms. The total spectrum has to be calculated for each of these arrays separately. Then, the system of incompletely specified Boolean functions should be processed by the following algorithm.

Algorithm 3: Spectral coefficient calculation for a system of arbitrary incompletely specified Boolean functions:

1) Represent each function in the system of Boolean functions by arrays of disjoint ON- and DC-cubes according to Algorithm 1.
2) Calculate the spectrum of an array of ON-cubes for each separate function $F[i]$ according to Algorithm 2.
3) Calculate the total spectrum $S_{TOT\,ON}$ of the system by using Properties 6.1 and 6.2.
4) Calculate the spectrum of an array of DC-cubes of each separate function $F[i]$ according to Algorithm 2.
5) Calculate the total spectrum $S_{TOT\,DC}$ of the system by using Properties 6.1 and 6.2.

Example 10: An example of the calculation of spectra $S_{TOT\,ON}$ and $S_{TOT\,DC}$ of a system of two incompletely specified Boolean functions ($b = 2$) having four variables is shown in Tables III and IV. The function $F[2]$ in this example is the same as the one used in Examples 1, 2, 3, and 9. The function $F[1]$ is taken from [13]. Both functions have no restriction in the choice of don't care minterms, therefore Algorithm 3 has to be performed. The original functions are presented in Fig. 4(a) (function $F[1]$) and Fig. 4(d) (function $F[2]$). The sets of ON-minterms that describe ON-cubes is presented in Fig. 4(b) and Fig. 4(e). The sets of dc-minterms are shown in Fig. 4(c) and Fig. 4(f). The corresponding arrays of disjoint ON- and DC-cubes are generated by Algorithm 1 (step 1). The execution of the second step of Algorithm 3 for ON-cubes is shown in the first two rows of Table III. The modified value of the spectrum of function $F[1]$ is shown in the third row of Table III (step 3). Since for the function $F[2]$ the modified value of spectrum is equal to the original one then this value is not repeated in the table. The total spectrum $S_{TOT\,ON}$ of this system of functions is the sum of rows one and three and is shown in the bottom row of Table III.

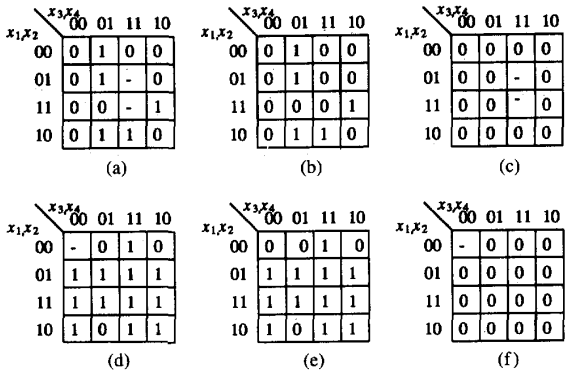


Fig. 4. Incompletely specified Boolean functions $F[1]$ and $F[2]$.

The execution of the fourth step of Algorithm 3 is shown in the first two rows of the Table IV. The modified value of the spectrum of function $F[1]$ is shown in the third row of Table IV (step 5). Since for the function $F[2]$ the modified value of spectrum is equal to the original one, this value is not repeated in the table. The total spectrum $S_{TOT\,DC}$ of this system of functions is the sum of rows one and three and is shown in the bottom row of Table IV.

A system of completely specified Boolean functions or incompletely specified Boolean functions with Restriction

1 can be represented by one spectrum. For this, Algorithm 3 can be simplified to Algorithm 4.

Algorithm 4: Spectral coefficients calculation for a system of an incompletely specified Boolean functions (with Restriction 1) or a system of completely specified Boolean functions.

1) Represent each function in the system of Boolean functions by arrays of disjoint ON- and DC-cubes according to Algorithm 1.
2) Calculate the spectrum of each separate function $F[i]$ according to Algorithm 2.
3) Calculate the total spectrum S_{TOT} of the system by using Properties 6.1 and 6.2.

VII. Implementation of the Algorithm for the Calculation of the Walsh Transform

The main problem of the implementation of the algorithm for the Walsh transform is the memory requirement for storing the whole spectrum. For an n-variable Boolean function the spectrum S has 2^n coefficients. Up to now, all other algorithms known to the authors have to keep the complete 2^n values of the spectral coefficients in the main computer memory. Thus, only Boolean functions with up to 18–20 literals could be processed [54]. Therefore, we have designed several algorithms for the generation of the spectral coefficients that do not keep all the coefficients in the computer memory at the same time. Since the trade-off exists between the execution speed and the area of the required memory, the concept of the transfer of control among the algorithms has been introduced. The user's options and the cooperation among the algorithms are shown in Fig. 5. The dashed arrows denote the options that can be selected by the user, while the continuous arrows denote the transfer of the control among the algorithms.

First, the user can choose options from one of three generations: of the Whole Spectrum, of Certain Orders of Coefficients (recall the definition of the order from Property 3.4), or of Some Spectral Coefficient. When the option of Some Spectral Coefficient is chosen, the coefficients are generated directly from the cubes. In the other cases, the user has to choose whether the orders of coefficients should be generated according to Algorithm A1 or A2. Then the program tries to allocate the necessary memory space for the required number of spectral coefficients nc according to the chosen algorithm. The value of nc is calculated by the formulas shown in Fig. 5, where n denotes the number of input variables of a Boolean function, and o the current order of coefficients. When the memory allocation fails then the successive algorithm having smaller memory requirements is automatically chosen (transition from Algorithm A1 through A2 till A2 or from Algorithm A2 to A3). When the coefficients of the next order are going to be generated the transition from Algorithm A3 to A2 is always possible and always tried by the program. The transition from Algorithm A2 to A1 is tried only if the Previous Order option has been chosen.

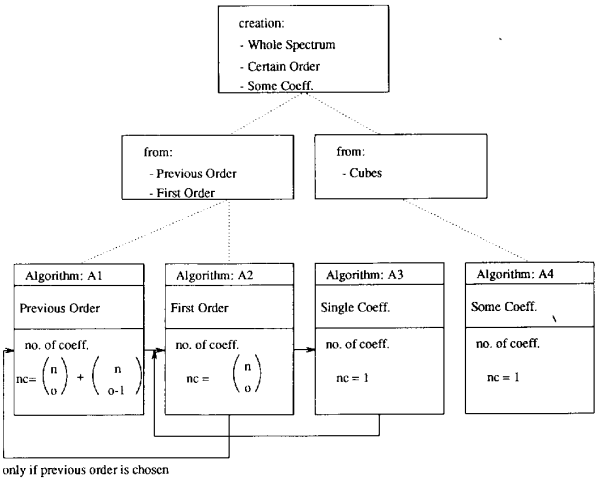


Fig. 5. Mutual relationships among the algorithms, the possible transitions and required memory size for each algorithm.

A. Algorithm A1: To Generate Indexes from the Previous Order

Algorithm A1 is optimized for the case when there are many cubes having many DC literals. This algorithm has to store in the memory two adjacent orders of spectral coefficients and the corresponding indexes due to the generation of the current order from the previous order.

The main part of Algorithm A1 is the generation of the indexes for the next order of spectral coefficients, since coefficients are generated in the Rademacher ordering. This part of the algorithm takes only such indexes of the previous orders for which the MSB (Most Significant Bit) is equal to 0, shift them to the left, and adds one. Next the part of the new order just generated is shifted again to the left, with the above restriction still valid. Thus, the next block is created. The procedure continues until the last index is generated. The details of the implementation of this procedure are described below.

Procedure: Generation of the indexes from the previous order:

1. do for indexes of the whole previous order
   if (MSB of index is 0)
   then: shift index to the left and add 1.
   else: go to next index.
2. do for each new generated block
   if (MSB of index is 0)
   then: shift index to the left.
   else: go to next index.

An example of this procedure is shown in Fig. 6. It is assumed, that the cubes have 5 literals and therefore the length of required indexes is also 5. The indexes of the third-order coefficients are generated from the second order. The circled areas mark the indexes having the MSB's equal to 0 on which the operations described by arrows are executed. Since $\binom{5}{2}$ is equal to $\binom{5}{3}$, then the number of indexes of the second and third orders is equal to 10.

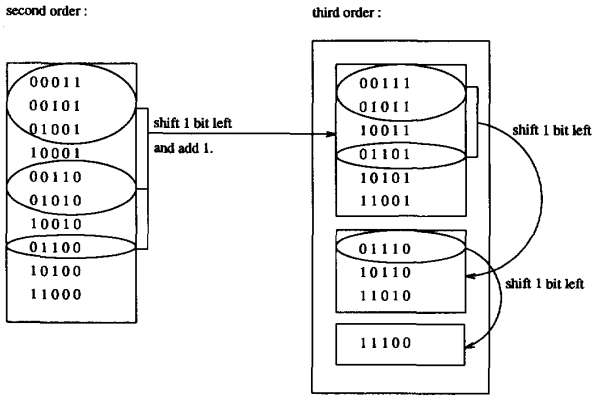


Fig. 6. Generation of third-order indexes of coefficients from the second order.

In the next part of Algorithm A1, the values of the spectral coefficients are generated by comparing the indexes to the cubes. To speed up the comparison (i.e., checking if the index of the spectral coefficient coincides with the cube), the notation of the currently processed cube is changed. The data about the cube is stored in two long variables: *valuex* and *value0*. The idea of the new notation is to make it the same as the notation for the indexes. In the variable *valuex*, 1s are stored only at the positions where the cube has *X*'s; otherwise the pattern is filled with 0's. In the variable *value0*, 0's are stored at the positions where the cube has 0s, otherwise the pattern is filled with 1s. If the intersection between the index of the spectral coefficient and the variable *valuex* is not empty, the value of the spectral coefficient must be zero. In the other case, the number of 1's in the intersection of the index of the spectral coefficient and the complement of the variable *value0* determines the sign of the value of the spectral coefficient.

Example 11: In Fig. 7, the *valuex* and the *value0* are shown for a certain cube. Because the intersection of *valuex* and the index is empty, the intersection between the index and the complement of *value0* has to be performed. Since the result of the second intersection has an odd number of 1s and the fifth order is odd, the sign of this coefficient (s_{14}) has to be minus.

B. Algorithm A2: To Generate Indexes from the First Order

Algorithm A2 is faster for the case when the cubes in the array have a small number of dc literals. It is a recursive algorithm to generate the indexes and values of the spectral coefficients of one order. The number of levels of recursions of the procedure is equal to the number of the current order. Each level of recursion changes one subindex (see the definition of a subindex in Property 3.4) of the index.

Example 12: This example demonstrates how the indexes of the third order for cubes with five literals are being generated. The indexes of the third order are determined by all permutations of three out of the five indexes from the first order (Fig. 8).

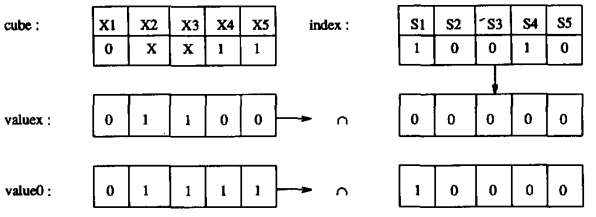


Fig. 7. Generation of the values of spectral coefficients.

Let us assume that the three variables $i[1]$, $i[2]$, and $i[3]$ hold the three subindexes of an index I of the third order spectral coefficient for some cube (e.g., $s_{124} \rightarrow i[1] := 1, i[2] := 2, i[3] := 4$). The numbers 1, 2, and 3 of $i[\,]$ indicate the level of the recursive procedure. Because the value of the spectral coefficient is the same as the values in the first order, only its sign has to be determined. It is calculated by multiplying the sign of the current spectral coefficient with the signs of the spectral coefficients of the first order, which are determined by the variables $i[1]$, $i[2]$, or $i[3]$. When the recursive procedure reaches the third level or executes the loop in the third level, a new spectral coefficient (its index and value) is stored. When the procedure is entered for the first time, the subindex $i[1]$ is set to 1, and the *sign* that has been determined by the order is multiplied by the sign of $s_{i[1]}$. The variable *indexI* is the index of the spectral coefficient $s_{i[1]}$. Because it is not yet in the third level, the procedure is called again. Then the subindex of the next level, which is always greater by 1 than the subindex of the previous level, is generated. This happens only after changing the level of the procedure. Thus, $i[2]$ is equal to 2. The *sign* is multiplied by the sign of $s_{i[2]}$, and the index of $s_{i[2]}$ is added to the *indexI*. Then, in the next level, $i[3] = i[2] + 1$ is set. The *sign* is multiplied by the sign of $s_{i[3]}$, and the index of $s_{i[3]}$ is added to *indexI*. Now, being in the third level, the first spectral coefficient of the third order is stored. The index of this spectral coefficient is equal to *indexI*, and its value is equal to the absolute value of any spectral coefficient from the first order multiplied by the variable *sign*. After that, the index of the spectral coefficient of the first order that has just been used is subtracted from the *indexI*, and the *sign* is multiplied by the coefficient's sign. Since $i[3]$ is not yet 5, it is incremented and the loop is done again. When $i[3] = 5$, this level is finished and procedure is back in level two. Then $i[2]$ is incremented and the procedure is called again. The execution of the recursive procedure continues until all three subindices $i[1]$, $i[2]$, and $i[3]$ have their highest possible values. All stages of this procedure are shown in Table V.

C. Algorithm A3: To Generate Order Step by Step

Algorithm A3 is called when the memory allocation for Algorithm A2 fails. Algorithm A3 needs only enough memory space for one single spectral coefficient to gen-

s_1	00001
s_2	00010
s_3	00100
s_4	01000
s_5	10000

Fig. 8. Indexes of the first-order coefficients.

TABLE V
GENERATION OF THE INDEXES OF THE THIRD ORDER BY THE RECURSIVE PROCEDURE

$i[1]$	$i[2]$	$i[3]$
1	2	3
		4
		5
	3	4
		5
	4	5
2	3	4
		5
	4	5
3	4	5

erate one complete order of spectral coefficients. This algorithm is similar to Algorithm A2. One difference is that no memory is allocated before calling the procedure to generate the index and the value for each spectral coefficient. This procedure is almost the same as the procedure for Algorithm A2. The second difference is that if one index is obtained, the value of the spectral coefficient for the whole array of cubes is immediately generated. This spectral coefficient is stored immediately on the hard disk, and the next spectral coefficients are calculated.

D. Algorithm A4: To Generate Certain Spectral Coefficients

In order to generate only certain spectral coefficients out of the whole spectrum, it is not necessary to create the indexes. Therefore Algorithm A4 does not use the long variables used in previously described algorithms to store the indexes. Since the implementation of the disjoint algorithm has also not been limited to a particular size of cubes, it is possible to generate separately spectral coefficients for cubes with an *arbitrary* number of literals. The literals necessary according to the given index are used directly in calculating the spectral coefficient in order to determine the sign of the value of this coefficient. The value itself is calculated according to the number of X's in the cube (denoted by the symbol p in Section V). For an array of cubes, it is simply done for each cube in turn, and the values are added to get the final value of the spectral coefficient.

VIII. EXECUTION OF WALSH TRANSFORM PROGRAM: A COMPLETE EXAMPLE

The array of disjoint cubes generated in Example 3 and used also as an example for Algorithm 2 (see Fig. 3) is the input data for the execution of the Walsh transform's program.

TABLE VI
GENERATION OF THE FIRST-ORDER AND DC SPECTRAL COEFFICIENTS

		S_0	S_1	S_2	S_3	S_4
$X1XX$	ON	0	0	16	0	0
$X011$	ON	12	0	12	4	4
$10X0$	ON	24	4	8	4	0
0000	DC	31	3	7	3	−1
		−9	3	7	3	−1

TABLE VII
GENERATION OF THE SECOND-ORDER SPECTRAL COEFFICIENTS

		S_{12}	S_{13}	S_{14}	S_{23}	S_{24}	S_{34}
$X1XX$	ON	0	0	0	0	0	0
$X011$	ON	0	0	0	4	4	−4
$10X0$	ON	4	0	4	4	0	−4
0000	DC	3	−1	3	3	−1	−5
		3	−1	3	3	−1	−5

It is not shown how the orders of spectral coefficients are generated because it has already been explained in Section VII (Fig. 5). However, it should be stressed that in Table II the spectrum for each cube is shown separately, while using either Algorithm A1 or A2 immediately adds the values of the spectral coefficients of a currently processed cube to the already calculated values corresponding to the previous cubes. By using Algorithm A1 or A2 only one complete order of the spectral coefficients can be created at a time. In the beginning, the coefficients of the first order as well as the DC coefficient (s_0) are generated, as shown in Table VI. After that, the next consecutive order is generated, as shown in Table VII. The same is done for the last two orders. The complete spectrum is shown in Table VIII.

IX. MEMORY AND TIME REQUIREMENTS FOR WALSH TRANSFORM PROGRAM

Table IX shows only the generation of the whole spectrum for up to 20 literals in each cube. It could be possible to do this for up to the program maximum, i.e., 32 literals. But even with only 20 literals one needs 3 Mbyte to store the complete spectrum on a hard disk. This problem can be partially overcome by using compression algorithms to store the spectrum, but it is inherent to the spectral methods that the number of coefficients grows exponentially.

To compare the processing time dependent on different arrays of cubes for the Sequent SYMMETRY 27 computer, Table IX is shown below.

The meaning of the abbreviations in Table IX is as follows (all values are in seconds):

first	indexes are generated from the first order (Algorithm A2),
previous	indexes are generated from the previous order (Algorithm A1),
u	elapsed user time,
s	elapsed system time.

TABLE VIII
COMPLETE RADEMACHER–WALSH S SPECTRUM OF FOUR-VARIABLE BOOLEAN FUNCTION

S_0	S_1	S_2	S_3	S_4	S_{12}	S_{13}	S_{14}	S_{23}	S_{24}	S_{34}	S_{123}	S_{124}	S_{134}	S_{234}	S_{1234}
−16	3	7	3	−1	3	−1	3	3	−1	−5	−1	3	−1	−5	−1

TABLE IX
EXECUTION TIMES

SYMMETRY 27					
Literals	Type	Previous		First	
16	*XX*	6.3u	0.7s	9.9u	1.8s
16	1*X*	8.9u	0.9s	27.7u	3.8s
18	*XX*	27.1u	3.1s	37.4u	2.8s
16	11	79.1u	6.0s	46.5u	7.8s
18	1*X*	35.1u	3.8s	106.1u	4.5s
20	*XX*	111.5u	21.3s	148.2u	32.4s
20	1*X*	137.8u	26.0s	435.2u	110.2s
18	11	339.5u	33.9s	182.7u	12.4s
20	11	1458.1u	69.9s	732.2u	144.0s

The first column of the table shows the number of literals per cube. Each array consists of ten cubes of the same type, where the type is determined in the second column of the table (XX-cube contains many DC-literals, 1X-cube contains some DC-literals, 11-cube contains no DC-literals). In order to obtain time data, a set of such examples has been tested. The time values are sorted according to their length.

Table IX shows one reason why different algorithms have been implemented for the generation of the whole spectrum. It has been illustrated that any no single algorithm, other than the one generating single coefficients, can give a solution for all cases while the combination of algorithms always gives the solution (the usage of the "worst case" single coefficient algorithm for every data would be inefficient).

The following conclusions can be derived from the obtained data.

- One can observe that the calculation of cubes with many X's is much faster than with a small number of X's.
- The algorithm that generates the spectral coefficients out of the previous order is up to three times faster for cubes having many X's, while the algorithm to generate the spectral coefficients out of the first order is up to two times faster for generating cubes that have few X's.
- By comparing the obtained results with the ones given in [51] (where the calculation of the spectrum for the function represented in the form of the truth table and of 18 literals took 382 s on VAX 11/780), one can observe that for the given cases our program is several times faster. A timing/synthesis comparison with SPECSYS is not possible since the detailed SPECSYS data have not been published. The SPECSYS program has not been made available to the authors.
- Our preprocessing algorithm to generate disjoint cubes takes only insignificant time (less than 1 s) for all tested cases. Therefore, the time presented in Table IX is the total processing time, which includes the time for the preprocessing. By contrast, the preprocessor for the algorithm of [51], [54] to create truth tables of Boolean functions takes a substantial amount of computer memory, and no time data has been published on it.

A. Memory Analysis

The memory requirement to calculate spectra is the most important factor. Because of that requirement, the existing algorithms according to Fast Transforms (Fast Algorithms) could compute only the spectrum for functions with up to 20 literals. The notation in this section is according to [56].

The basic memory is the same for the introduced Algorithms A1, A2, and A3 for the calculation of the Walsh spectrum, and has to store the array of cubes, the program itself, and the necessary number of coefficients.

The maximal memory requirement to store the array of cubes is given by the number of ON- or ON/DC minterms which specify the function. Usually, the function is represented by cubes that are larger than minterms. Hence, $cu \leq 2^l$, where cu is the number of cubes and l denotes the number of literals. It was shown in [3] that the number of cubes cu is much smaller than 2^l for practical functions. In the implementation, eight literals are stored in one integer variable. Hence, the following memory is necessary to hold the array of cubes:

$$cu \times \begin{cases} 2 \text{ bytes} & 1 \leq 8 \\ 4 \text{ bytes} & 8 \leq l \leq 16 \\ 6 \text{ bytes} & 16 \leq l \leq 24 \\ 8 \text{ bytes} & 24 \leq l \leq 32. \end{cases}$$

The program itself uses about 50 kbytes of computer memory. Fig. 9(a) compares the number of spectral coefficients (nc) that have to be kept in the computer memory to generate a complete spectrum for three different algorithms (A1, A2, A3) and for the Fast Algorithms. The formulas describing nc that have to be kept in the computer memory are shown in Fig. 5. In the case of Fast Algorithms, $nc = 2^l$ is needed (Fig. 9(a)). To store the value of one spectral coefficient 4 bytes are used because each value is stored in a long variable.

If there is not enough memory left during an execution of the program, the Algorithm A3 is used. This means that only the memory for one single spectral coefficient is necessary. Thus, the program is only limited by the mem-

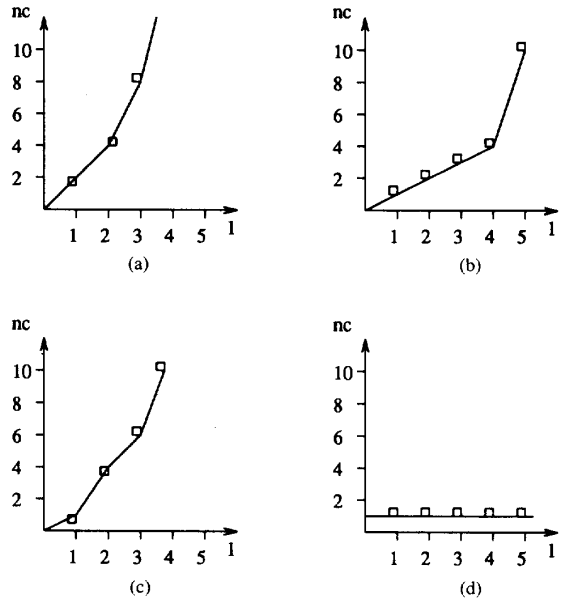


Fig. 9. Memory requirements ($nc \times 4$ bytes). (a) Existing algorithms. (b) Algorithm A1. (c) Algorithm A2. (d) Algorithm A3. l: number of literals; nc: number of coefficients.

necessary to store the array of cubes and the program itself. The memory requirements $m(l)$ are given by

$$m(l) = O\,(50 \text{ kbytes} + (2^l - 1) \times 8 \text{ bytes}) \tag{30}$$

$$m(l) = o\left(50 \text{ kbytes} + (2^l - 1) \times 8 \text{ bytes} + \left[\binom{n}{\frac{n}{2}} + \binom{n}{\frac{n}{2} - 1}\right] \times 4 \text{ bytes}\right) \tag{31}$$

$$m(l) = o\left(50 \text{ kbytes} + (2^l - 1) \times 8 \text{ bytes} + \left[\binom{n}{\frac{n+1}{2}} + \binom{n}{\frac{n+1}{2} - 1}\right] \times 4 \text{ bytes}\right) \tag{32}$$

where 50 kbytes is the memory required for the program and $(2^l - 1)$ means that the function is represented by all but one minterm. Formula (31) applies for even n, (32) for odd n.

B. Time Analysis

To estimate the execution time for the implementation of the Walsh transformation, Algorithm A3, which represents the upper bound, has been used. This is a straightforward algorithm that calculates every spectral coefficient directly from the cube representation without reference to its contribution to the current spectral coefficient calculated. Thus, the time increases according to the number of literals (l) in the function as well as the number of cubes (c):

$$g(l, c) = const \times 2^l \times c \tag{33}$$

where $g(l, c)$ gives the time (in seconds) depending on the number of literals and cubes, and *const* is the necessary time to calculate the value of one spectral coefficient for a cube consisting of one literal. Hence, the function $g(l, c)$ gives an upper bound for the actual processing time $f(l, c)$:

$$f(l, c) = o(g(l, c)) \tag{34}$$

To evaluate the constant *const*, the time values $g(l, c)$ shown in Table X have been used.

TABLE X

	10	const	11	const
1	0.3u	0.000 2929	0.7u	0.000 3418
2	0.5u	0.000 2441	1.1u	0.000 2686
5	1.2u	0.000 2344	2.5u	0.000 2441
10	2.3u	0.000 2246	4.9u	0.000 2393
50	11.5u	0.000 2246	23.3u	0.000 2275

The first column in the table indicates the number of disjoint cubes c, and the first row the number of literals l for which the spectrum has been calculated. In the second and third columns, the value of *const* is given. The obtained value is $0.000\,2246 < const < 0.000\,3418$, where for a small number of cubes the calculation of the index of the spectral coefficient itself has a larger contribution to *const*. Then, the approximate value for $const = 2^{-12}$. Thus, the upper bound time for the implemented algorithm is:

$$g(l, c) = 2^{l-12} \times c. \tag{35}$$

The advantage of the Algorithm A1 and A2 in the processing time is shown in Fig. 10.

For the graphic in Fig. 10, the better time of either Algorithm A2 or A1 has been used. The processing time for the different kinds of cubes can be given as

$$f_{xx}(l, c) = \Theta_1(f_{1x}(l, c)) = \Theta_2(f_{11}(l, c)) \tag{36}$$

where the subscript of f denotes the type of cubes and f was defined in (34).

Because the basic part of Algorithms A1, A2, and A3 is similar and the algorithms are performed for each type of cubes, their relative processing times are distinguished only by a constant. Thus, function Θ depends only on the number of DC-literals in the cubes. This is caused by the fact that this number determines how many operations the algorithms must perform.

X. Conclusion and Future Work

The essential relationships between classical and spectral methods used in the design of digital circuits have been stated. Based on these relations, new algorithms for

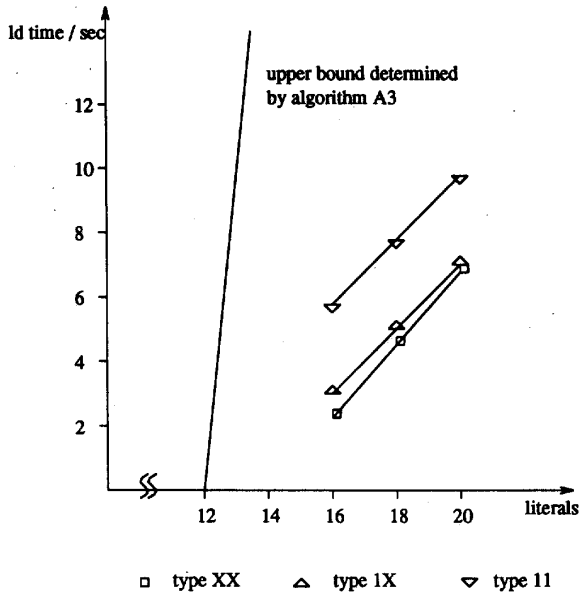


Fig. 10. Time comparison of Algorithm A3 versus A2 and A1.

the generation of spectral coefficients for both S and R spectra for completely as well as incompletely specified Boolean functions have been shown. The graphical method for the calculation of spectral coefficients directly from a Karnaugh map is a powerful and efficient tool for functions with the number of variables less than or equal to six. Alternative formulas for the calculation of s_I and r_I spectral coefficients have also been presented. Such detailed interpretations of R and S spectra of Boolean functions are important not only from the point of view of analysis and synthesis of digital systems, but also for the generation of tests and design for testability.

A new, efficient algorithm and its implementation for the generation of spectral coefficients have been described. The computer method for performing this algorithm has been implemented in the DIADES automation system developed at Portland State University. The SPECSYS (for SPECtral SYnthesis System) developed at Drexel University on VAX 11/780 uses the Fast Walsh Transform for the calculation of the spectrum and can only process Boolean functions having up to 20 input variables [51], [54]. The DIADES program has no limit on the number of input variables of Boolean functions and it applies the methods described in this article for the generation of spectral coefficients of Boolean functions.

The authors of SPECSYS program [54] have encountered the disadvantages listed below while using Fast algorithms for calculating Walsh spectra. By implementing the approach described in this paper, the DIADES system successfully overcomes most of these disadvantages.

The first disadvantage of SPECSYS, one which causes the use of excessive computer memory, is that the computation of the Walsh spectrum requires the representation of the Boolean function in the form of a truth table composed of minterms. In DIADES, by contrast, the spectrum is generated directly from the reduced representation of Boolean functions (arrays of disjoint cubes) [2], [8], [46]. 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, especially the fact that for practical functions the number of disjoint cubes is much smaller than the number of minterms, result from [3]. It is also evident from Table I that the number of disjoint cubes of the presented functions is much smaller than the number of their minterms.

The second disadvantage of SPECSYS is that all spectral coefficients must be calculated at once. In our approach, the entire spectrum, if required, can be computed incrementally for groups of coefficients. Therefore our computer method is very efficient for the calculation of only the few selected spectral coefficients, which is all that is needed in many synthesis methods [4], [10], [21]–[26], [31]–[35], [38], [48], [49], [53], [55].

The third disadvantage of SPECSYS is that it can use only completely specified Boolean functions. DIADES operates on systems of both completely and incompletely specified Boolean functions. The other advantages of the algorithms implemented in DIADES have been described in the article. The only drawback of the DIADES approach is the exponential growth of hard disk storage requirements with the increase in the number of coefficients. This is inherent to the nature of the problem. In SPECSYS the storage requirements are even worse, since it uses only internal memory. The implementation of the described algorithm allows the calculation of the spectrum for completely and incompletely specified Boolean functions having up to 32 variables. Since our system can calculate coefficients either by groups or separately, in the worst case it requires only enough memory to hold the first order spectral coefficients. The $n \leq 32$ constraint refers to the generation of either a complete order or the whole spectrum. It should be noticed, however, that even for the cases when n is limited, it can be increased when a list structure that describes the indexes (see Property 3.5) is created. With such a list, the spectrum of a Boolean function having an arbitrary number of variables can be calculated, the only limitation being the memory available on the hard disk. When the coefficients are calculated separately, even with the current implementation, there is no limit on n since the coefficients can be stored in groups on the hard disk. This is, however, traded off for the increased processing time. When the whole spectrum is not required, the algorithm can calculate chosen spectral coefficients for Boolean functions of an arbitrary number of variables. The results presented in the article show that our system is currently the fastest and most flexible spectral synthesis system designed.

Computer algorithms similar to the one presented in this article have already been developed for the newly introduced Generalized Arithmetic and Adding transforms [11], Walsh-type transforms of completely and incom-

pletely specified multiple-valued input binary functions [12], and the Reed–Muller transform [17]. The decomposition and linearization methods for the spectra of systems of incompletely specified Boolean functions with Restriction 1 are presented in [25]. A suggested goal for future research is the development of new decomposition and linearization methods for systems of arbitrary Boolean functions. One possible approach to this problem is to apply the known methods of [25] to S_{TOTON} and S_{TOTDC} in turn. More investigations are needed in this area.

The fundamental formulas presented in Section II are very useful when used for the investigation of new transforms, relations among various transforms, and the relationships between classical logic analysis and synthesis methods and spectral methods, especially when one attempts to explain the meaning of new concepts using well-established notions. Understanding these principles gives us the working tool to translate in both directions the notions of classical and spectral theories, design of new hardware realizations for various transforms (including also those that are different from Walsh type), testing procedures, and synthesis methods.

The interpretations and algorithms, analogous to those presented in Section II, for only s_l and r_l spectral coefficients can be derived in a similar way for the weighted sum of the spectral coefficients as well as for the autocorrelation function of the Boolean function [34], [48], [49]. Both these parameters of Boolean functions have been found very useful in testing. For example, the testing of programmable logic arrays by the weighted sum of spectral coefficients provides 100% coverage of all single stuck-at faults and very high coverage of multiple-faults [48].

The research summarized here will have an impact of the application of Boolean and multiple-valued input logic not only in the synthesis, analysis, and testing of digital circuits but in areas of pattern recognition and signal processing as well. The goal of future research is to develop new decomposition methods for systems of incompletely specified Boolean functions based on the representation of the Rademacher–Walsh spectrum presented. The properties of such decompositions make them very suitable for design using FPGA's [54]. A major advantage of the approach to Walsh-spectrum calculation presented here is its convenience for computer implementation and its ability to yield solutions to problems of very high dimensions.

Acknowledgment

The authors wish to especially thank the anonymous referees for numerous helpful comments.

References

[1] K. G. Beauchamp, *Applications of Walsh and Related Functions*. New York: Academic, 1984.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA: Kluwer Academic, 1985.

[3] M. J. Ciesielski, S. Yang, and M. A. Perkowski, "Minimization of multiple-valued logic based on graph coloring," Tech. Rep. TR-CSE-90-13, Dep. Elect. Comp. Eng., Univ. of Massachusetts, Amherst, Sept. 1990. (Earlier version of this paper appeared as: "Multiple-valued minimization based on graph coloring," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 262–265, 1989.)

[4] T. Damarla and M. G. Karpovsky, "Reed-Muller spectral techniques for fault detection," *IEEE Trans. Comput.*, vol. 38, pp. 788–797, June 1989.

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

[6] R. C. Debnath and A. K. Karmakar, "Method for finding Rademacher–Walsh spectral coefficients of Boolean functions," *Int. J. Electron.*, vol. 60, pp. 245–250, Feb. 1986.

[7] E. Detjens, "FPGA devices require FPGA-specific synthesis tools," *Computer Design*, p. 124, Nov. 1990.

[8] D. L. Dietmayer, *Logic Design of Digital Systems*. Boston, MA: Allyn and Bacon, 1978.

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

[10] E. Eris and J. C. Muzio, "Spectral testing of circuit realizations based on linearizations," *Proc. IEE Comput. & Digital Tech.*, vol. 133, pp. 73–78, 1986.

[11] B. J. Falkowski and M. A. Perkowski, "One more method for the calculation of Hadamard-Walsh spectrum for completely and incompletely specified Boolean functions," *Int. J. Electron.*, vol. 69, no. 5, pp. 595–602, Nov. 1990.

[12] ——, "Algorithms for the calculation of Hadamard-Walsh spectrum for completely and incompletely specified Boolean functions," in *Proc. 9th IEEE Int. Phoenix Conf. on Computers and Communications*, Scottsdale, AZ, pp. 868–869, Mar. 1990.

[13] ——, "Essential relations between classical and spectral approaches to analysis, synthesis, and testing of completely and incompletely specified Boolean functions," in *Proc. 23rd IEEE Int. Symp. on Circuits and Systems*, New Orleans, LA, pp. 1656–1659, May 1990.

[14] ——, "A family of all essential radix-2 addition/subtraction multipolarity transforms: algorithms and interpretations in Boolean domain," in *Proc. 23rd IEEE Int. Symp. on Circuits and Systems*, New Orleans, LA, pp. 2913–2916, May 1990.

[15] ——, "Algorithm and architecture for Gray code ordered fast Walsh transform," in *Proc. 23rd IEEE Int. Symp. on Circuits and Systems*, New Orleans, LA, pp. 1596–1599, May 1990.

[16] ——, "Walsh type transforms for completely and incompletely specified multiple-valued input binary functions," in *Proc. 20th IEEE Int. Symp. on Multiple-Valued Logic*, Charlotte, NC, pp. 75–82, May 1990.

[17] ——, "On the calculation of generalized Reed-Muller canonical expressions from disjoint cube representation of Boolean functions," in *Proc. 33rd Midwest Symp. on Circuits and Systems*, Calgary, Alberta, pp. 1131–1134, Aug. 1990.

[18] B. J. Falkowski, I. Schäfer, and M. A. Perkowski, "A fast computer algorithm for the generation of disjoint cubes for completely and incompletely specified Boolean functions," in *Proc. 33rd Midwest Symp. on Circuits and Systems*, Calgary, Alta., Canada, pp. 1119–1122, Aug. 1990.

[19] M. Helliwell and M. A. Perkowski, "A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms," in *Proc. 25th ACM/IEEE Design Automation Conf.*, Anaheim, CA, pp. 427–432, Jun. 1988.

[20] D. Green, *Modern Logic Design*. Wokingham, MA: Addison-Wesley, 1986.

[21] T. Hsiao and S. C. Seth, "An analysis of the use of Rademacher-Walsh spectrum in compact testing," *IEEE Trans. Comput.*, vol. C-33, pp. 934–938, Oct. 1984.

[22] S. L. Hurst, *The Logical Processing of Digital Signals*. New York: Crane-Russak, 1978.

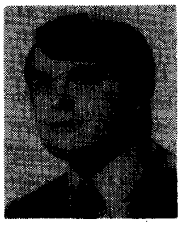
[23] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*. London, U.K.: Academic, 1985.

[24] S. L. Hurst, "Use of linearization and spectral techniques in input and output compaction testing of digital networks," in *Proc. IEE Comput. & Digital Tech.*, vol. 136, pp. 48–56, Jan. 1989.

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

[26] M. G. Karpovsky, ed., *Spectral Techniques and Fault Detection*. Orlando, FL, Academic, 1985.

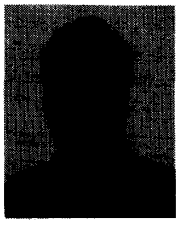
[27] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.
[28] H. Landman, "Logic synthesis at Sun," IEEE Conference Paper, CH2686-4/89/0000/0469, 1989.
[29] R. Lechner, "Harmonic analysis of switching functions," in *Recent Developments in Switching Theory*. A. Mukhopadhyay, Ed., New York: Academic, 1971.
[30] A. M. Lloyd, "Spectral addition techniques for the synthesis of multivariable logic networks," *Proc. IEE Comput. & Digital Tech.*, vol. 125, pp. 152–164, 1978.
[31] A. M. Lloyd, "Design of multiplexer universal-logic-module networks using spectral techniques," *Proc. IEE Comput. & Digital Tech.*, vol. 127, pp. 31–36, 1980.
[32] P. K. Lui and J. C. Muzio, "Spectral signature testing of multiple stuck-at faults in irredundant combinational networks," *IEEE Trans. Comput.*, vol. C-35, pp. 1088–1092, Dec. 1986.
[33] D. M. Miller and J. C. Muzio, "Spectral fault signatures for single stuck-at faults in combinational networks," *IEEE Trans. Comput.*, vol. C-33, pp. 765–768, Aug. 1984.
[34] D. M. Miller, ed., *Developments in Integrated Circuit Testing*. London, UK: Academic, 1987.
[35] J. C. Muzio and S. L. Hurst, "The computation of complete and reduced sets of orthogonal spectral coefficients for logic design and pattern recognition purposes," *Comput. & Elect. Eng.*, vol. 5, pp. 231–249, 1978.
[36] J. C. Muzio, "Composite spectra and the analysis of switching circuits," *IEEE Trans. Comput.*, vol. C-29, pp. 750–753, 1980.
[37] J. C. Muzio, D. M. Miller, and S. L. Hurst, "Number of spectral coefficients necessary to identify a class of Boolean functions," *Electron. Lett.*, vol. 25, pp. 577–578, 1982.
[38] J. C. Muzio and D. M. Miller, "Spectral fault signatures for internally unate combinational networks," *IEEE Trans. Comput.*, vol. C-32, pp. 1058–1062, 1983.
[39] L. Nguyen, M. Perkowski, and N. Goldstein, "PALMINI-fast Boolean minimizer for personal computers," in *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 615–621, 1987.
[40] M. A. Perkowski, M. Driscoll, J. Liu, D. Smith, J. Brown, L. Yang, A. Shamsapour, M. Helliwell, B. Falkowski, and A. Sarabi, "Integration of logic synthesis and high-level synthesis into the Diades design automation system," in *Proc. 22nd IEEE Int. Symp. on Circuits & Systems*, Portland, OR, pp. 748–751, 1989.
[41] M. Perkowski, M. Helliwell, and P. Wu, "Minimization of multiple-valued input multi-output mixed-radix exclusive sums of products for incompletely specified Boolean functions," in *Proc. 19th Int. Symp. on Multiple-Valued Logic*, pp. 256–263, 1989.
[42] M. A. Perkowski, P. Dysko, and B. J. Falkowski, "Two learning methods for a tree-search combinatorial optimizer," in *Proc. 9th IEEE Int. Phoenix Conf. on Computers & Communications*, Scottsdale, AZ, pp. 606–613, Mar. 1990.
[43] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Commun. ACM*, vol. 44, pp. 668–676, June 1990.
[44] S. M. Reddy, "Easily testable realizations for logic functions," *IEEE Trans. Comput.*, vol. C-21, pp. 1183–1188, Nov. 1972.
[45] B. R. K. Reddy and A. L. Pai, "Reed-Muller transform image coding," *Computer Vision, Graphics, and Image Processing*, vol. 42, pp. 48–61, 1988.
[46] J. P. Roth, *Computer Logic, Testing and Verification*. Potomac, MD: Computer Science Press, 1980.
[47] J. M. Sanchez, J. Ballesteros, and A. Vaquero, "Study of the complexity of an algorithm to derive the complement of a binary function," *Int. J. Electron.*, vol. 66, pp. 245–250, 1989.
[48] M. Serra and J. C. Muzio, "Testing Programmable Logic Arrays by sum of syndromes," *IEEE Trans. Comput.*, vol. C-36, pp. 1097–1101, Sept. 1987.
[49] M. Serra and J. C. Muzio, "Space compaction for multiple-output circuits," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 1105–1113, Oct. 1988.
[50] V. H. Tokmen, "Disjoint decomposability of multi-valued functions by spectral means," in *Proc. IEEE Int. Symp. on Multiple-Valued Logic*, pp. 88–93, 1980.
[51] E. A. Trachtenberg and D. Varma, "A design automation system for spectral logic synthesis," in *Proc. Int. Workshop on Logic Synthesis*, Research Triangle Park, NC, May 12–15, 1987.
[52] E. A. Trachtenberg, "Designing standard computer components using spectral techniques," in *Proc. IEEE Int. Conf. on Comp. Design: VLSI in Computers & Processors*, pp. 630–633, 1987.
[53] D. Varma and E. A. Trachtenberg, "A fast algorithm for the optimal state assignment of large finite state machines," in *Proc. IEEE Int. Conf. on Computer-Aided Design*, Santa Clara, CA, pp. 152–155, 1988.
[54] ——, "Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 901–916, Aug. 1989.
[55] ——, "On the estimation of logic complexity for design automation applications," in *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 368–371, Cambridge, MA, Sept. 17–19, 1990.
[56] H. S. Wilf, *Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1986.



Bogdan J. Falkowski (S'88–M'90) received the M.S.E.E. degree from Technical University Warsaw, Poland, and the Ph.D. degree from Portland, OR, State University.

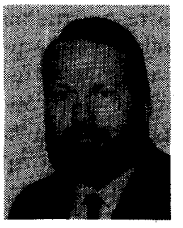
His industrial experience includes research and development positions at several companies from 1978 to 1986. He then joined the Electrical and Computer Engineering Department at Portland State University. In 1992, he became a senior lecturer in the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. His research interests include VLSI systems and design, switching circuits, testing, design of algorithms, design automation, digital signal, and image processing. He specializes in the design of digital circuits with the use of spectral methods and has published 15 articles in this area.

Dr. Falkowski is a member of Eta Kappa Nu and Pi Beta Upsilon.



Ingo Schäfer received the M.S. degree in electrical engineering from Portland State University, OR, in 1990. He is currently working toward the Ph.D. degree.

His research interests include logic synthesis, signal and image processing, and harmonic analysis of switching functions.



Marek A. Perkowski (M'84) was born in Warsaw, Poland, on October 6, 1946. He received the M.S. degree in electronics (automatic control) in 1970, and the Ph.D. degree in automatics (digital systems) in 1980, from Technical University of Warsaw.

He was an Assistant Professor at Technical University of Warsaw from 1980 to 1981, a Visiting Assistant Professor at the University of Minnesota from 1981 to 1983 and, since 1983, an Associate Professor of Electrical and Computer Engineering at Portland State University.

He had summer professor positions with GTE Labs, Intel Scientific Computers, and Sharp Microelectronics Technology, and is a consultant to Cypress Semiconductor Corp. He was a co-author of three books, five book chapters, and over 100 technical articles in design automation, computer architecture, Artificial Intelligence, Image Processing, and robotics. His recent research interests include designing Finite State Machines in Field Programmable Gate Arrays, and building a computer system for ovulation prediction.

Dr. Perkowski is a founding member of Polish Informatics Society.