CALCULATION OF THE RADEMACHER-WALSH SPECTRUM FROM A REDUCED REPRESENTATION OF BOOLEAN FUNCTIONS

Bogdan J. Falkowski, Ingo Schäfer +, Marek A. Perkowski +
Nanyang Technological University, School of Electrical and Electronic Eng., Singapore 2263.
+Portland State University, Department of Electrical Eng., Portland, OR 97207-751

ABSTRACT

A theory has been developed to calculate the Rademacher-Walsh transform from a reduced representation (disjoint cubes) of incompletely specified Boolean functions. 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 program for the algorithms uses 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) for the calculation of Rademacher-Walsh transform known to us. 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.

1. INTRODUCTION

In digital logic design, spectral techniques (Walsh transforms) have been used for more than thirty years. They have been applied to Boolean function classification, disjoint decomposition, parallel and serial linear decomposition, spectral translation synthesis (extraction of linear pre- and post-filters), multiplexer synthesis, prime implicant extraction by spectral summation, threshold logic synthesis, evaluation of logic complexity, and state assignment [1, 2, 9]. Spectral methods for testing of logical networks by verification of the coefficients in the spectrum have been developed [9]. 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 [9]. 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.

Two design automation systems have used spectral methods as the tool for designing of digital circuits [12-16]. 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, 9]. 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 [13]. The DIADES design automation system developed by us [12] 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.

A high demand exists for the methods that produce circuit realizations with EXOR gates [4, 10]. "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 PLDs." - quoted from [4]. "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 [10]. The decomposition of Boolean functions with both pre- and post-linear parts by spectral means allows for the realization of highly testable circuits for Field-Programmable Gate Arrays. Currently only spectral methods allow for this kind of decomposition [9, 15].

In this article, the main emphasis is laid 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, 9]. The ordering of Walsh transforms describes the sequence in which Walsh functions are placed in the transform matrix. There exist two Rademacher-Walsh spectra of Boolean functions known in the literature under the names of R and S spectrum [6, 7, 9]. In the following material, both 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) [1, 9]. 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.

In this article a *new method* for the calculation of the Rademacher-Walsh spectrum of incompletely specified Boolean functions is shown. The method presented 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

0-8186-2780-8/92 $3.00 © 1992 IEEE

described in [7] this article uses mainly the S spectrum.

This article 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 algorithm presented in this article this obstacle has been overcome.

2. AN ARRAY METHOD FOR THE CALCULATION OF A 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 [9, 11]. When the implicants are not mutually disjoint this algorithm requires 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 [2, 5, 8] one can apply the existing algorithm without having to perform additional 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 functions is proposed.

In what follows the properties of the existing algorithm are rewritten in the 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 are presented.

Definition 2.1: 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 2.1: Consider the cube $1X00$. 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 2.2: 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 2.2: Consider Table 1 representing the array method of calculating spectral coefficients. 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. 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 [7]. Similarly the spectrum of each don't care minterm is given by $s_{DC\,0} = 2^n - 1$, and all the remaining $2^{n-1} - 1$ coefficients are equal to ± 1 [7]. The symbols $s_{DC\,I}$ denote spectral coefficients corresponding to DC- cubes (when $I = 0$, the symbol $s_{DC\,0}$ denotes a direct current spectral coefficient corresponding to a DC- cube).

Cubes of degree m have the following properties.

Property 2.1: 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:

$$s_0 \text{ in full } n\text{-space} = 2^n - 2 \times 2^p \quad (1)$$

$$s_I \text{ in full } n\text{-space} = s_I \text{ in } m\text{-space} \times 2^p \quad (2)$$

where $I \neq 0$.

Property 2.2: The contribution of a DC- cube of degree m to the full n-space spectrum of function F is related as follows:

$$s_{DC\,0} \text{ in full } n\text{-space} = 2^{n-1} - 2^p \quad (3)$$

$$s_{DC\,I} \text{ in full } n\text{-space} = s_{DC\,I} \text{ in } m\text{-space} \times 2^p \quad (4)$$

where $I \neq 0$.

Equations (2) and (4) determine the absolute values of those partial spectral coefficient s_I that are not equal to zero for a given cube. Properties 2.3 - 2.5 determine the signs of the partial spectral coefficients, and if some of them are equal to zero.

Example 2.3: Consider again Table 1. 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 (1). The absolute values of those partial spectral coefficients s_I that are not equal to zero are calculated according to (2) 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 (3). The absolute values of those partial spectral coefficients s_I that are not equal to zero are calculated according to (4) 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.

Property 2.3: 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.

Property 2.4: 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 subindices i, j, k, etc. is equal to 0.

Property 2.5: 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 2.4: Consider again Table 1. The variable x_3 is denoted by symbol X in the cube $10X0$. Then, by Property 2.3 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 2.5, 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 2.4 and Property 2.5, 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:

Property 2.6: 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.

Property 2.7: The signs of all even order coefficients are given by multiplying the signs of the related first order coefficients by -1.

Property 2.8: The signs of all odd order coefficients are given by multiplying the signs of the related first order coefficients.

Example 2.5: Consider again Table 1. In the ON- cube $10X0$ the variable x_1 is in affirmation, while the variables x_2 and x_4 are in negation. Therefore, by Property 2.4 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 2.7. The sign of the even order partial spectral coefficient s_{12} of cube $10X0$ is positive, since this 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 2.8. The signs of the partial spectral coefficient s_{124} of the same cube is positive since it is determined according to Property 2.8 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: 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 the sign of the contribution of this cube to the full n-space spectrum according to the properties described previously.
2. 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 $-(k-1) \times 2^n$, where 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 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 2.6: An example of the calculation of the S spectrum for the four variable incompletely specified Boolean function is shown in Table 1. The values and signs of all the partial spectral coefficients for this function are determined by Properties 2.1 - 2.8.

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 then the steps 3 and 4 are not performed. The resulting spectrum is shown at the bottom row of Table 1.

3. 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 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 [17]. Therefore, several algorithms for the generation of the spectral coefficients have been designed by us that do not keep all the coefficients at the same time in the computer memory. 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. 1. 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 one of three generation's options: of the Whole Spectrum, of Certain Orders of Coefficients, 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. 1, 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 A1 through A2 till A3 or from A2 to A3). When the coefficients of the next order are going to be generated then the transition from A3 to A2 is always possible and tried by the program. The transition from A2 to A1 is tried only if the Previous Order option has been chosen.

3.1. Algorithm A1 to generate the indices from previous order

The 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 indices due to the generation of the current order from the previous order.

The main part of the algorithm A1 is the generation of the indices for the next order of spectral coefficients since coefficients are generated in the Rademacher ordering. This part of the algorithm takes only such indices of the previous orders for which the MSB (Most Significant Bit) is equal to 0, shifts them to the left, and adds one. Now, the just generated part of the new order is shifted again to the left with the above restriction still valid. Thus, the next block is created. The procedure continues till the generation of the last index. In the next part of the algorithm A1, the values of the spectral coefficients are generated by comparing the indices to the cubes.

3.2. Algorithm A2 to generate the indices from the first order

The 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 indices 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.

3.3. Algorithm A3 to generate order step by step

The algorithm A3 is called when the memory allocation for the algorithm A2 fails. It has the feature, that it needs only memory space for one single spectral coefficient to generate 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 the algorithm A2. The second difference is, that if one index is obtained, then 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.

3.4. 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 indices. Therefore the algorithm does not use the long variables that are used in previously described algorithms to store the indices. Since the implementation of the disjoint algorithm has also not been limited to a particular size of cubes then it is possible to generate separately spectral coefficients for cubes with an *arbitrary* number of literals. The needed literals according to the given index are directly taken to calculate 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 *Xs* in the cube (denoted by the symbol *p* in Section 2). 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.

4. MEMORY AND TIME REQUIREMENTS FOR WALSH TRANSFORM PROGRAM

In Table 2 only the generation of the whole spectrum for up to 20 literals in each cube is shown. 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 3Mbyte 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 2 is shown below.

Meaning of the abbreviations in the Table 2 is as follows (all values are in seconds):

- first indices are generated from the first order (algorithm A2),
- previous indices are generated from the previous order (algorithm A1),
- u elapsed user time,
- s elapsed system time.

In the first column of the table the number of literals per cube is shown. 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-literal). In order to obtain time data a set of such examples has been tested. The time values are sorted according to their length.

The above table shows one reason why different algorithms have been implemented for the generation of the whole spectrum. It has been illustrated that any particular algorithm, other then the one generating single coefficients, cannot give a solution for all cases while the combination of algorithms gave the solution always (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 a lot of Xs is much faster than with small number of Xs.
- The algorithm which generates the spectral coefficients out of the previous order is up to 3 times faster for cubes having many Xs. Where the algorithm to generate the spectral coefficients out of the first order is up to 2 times faster for generating cubes that have few Xs.
- By comparing the obtained results with the ones given in [13] (where the calculation of the spectrum for the function represented in the form of the truth table and of 18 literals took 382 seconds 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 [8] takes only insignificant time (less than 1 second) for all tested cases. Therefore, the time presented in Table 2 is the total processing time which includes the time for the preprocessing. In contrary, the preprocessor for the algorithm of [13, 15] to create truth tables of Boolean functions takes a lot of computer memory and no time data has been published on it.

4.1. Memory Analysis

The memory requirement to calculate spectra is the most important factor. Because of that requirement, the existing algorithms according to Fast Transforms could compute only the spectrum for functions with up to 20 literals.

The basic memory for the introduced algorithms A1, A2, and A3 for the calculation of Walsh spectrum is the same and has to store the array of cubes, keep the program itself, and store the necessary number of coefficients. The maximal memory requirement to store the array of cubes in the memory 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. In the implementation eight literals are stored in one integer variable. Hence, the following memory is necessary to hold the array of cubes.

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

If there is not enough memory left to store the complete spectrum 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 necessary memory to store the array of cubes and the program itself. The memory requirements $m(l)$ for the algorithm A3 are given by:

$$m(l) = O((2^l - 1) \times 8\ bytes) \tag{5}$$

$$m(l) = o((2^l - 1) \times 8\ bytes + \left[\left[\frac{n}{\frac{n}{2}}\right] + \left[\frac{n}{\frac{n}{2}-1}\right]\right] \times 4\ bytes) \tag{6}$$

$$m(l) = o((2^l - 1) \times 8\ bytes + \left[\left[\frac{n}{\frac{n+1}{2}}\right] + \left[\frac{n}{\frac{n+1}{2}-1}\right]\right] \times 4\ bytes) \tag{7}$$

where $(2^l - 1)$ means that the function is represented by all but one minterm. Formula (6) applies for even n, formula (7) for odd n.

5. CONCLUSION

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. 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 [13, 15]. 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 DIADES system overcomes the following disadvantages found in the SPECSYS program [15]. First, in DIADES the spectrum is generated directly from the reduced representation of Boolean functions (arrays of disjoint cubes) [2, 5, 8] rather than from the truth table (minterms). Second, 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 what is needed in many synthesis methods [1, 9, 11].

Third, 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's approach is the exponential growth of hard disk storage requirements with the increase of the number of coefficients. This is inherent to the nature of the problem. In SPECSYS the storage requirements are even worse since internal memory is all that is used. 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 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, however, be noticed that even for the cases when n is limited, it can be increased when a list structure that describes the indices is created. The results presented in the article show that our system is currently the fastest and most flexible spectral synthesis system designed.

The goal of future research is to develop new decomposition methods for systems of incompletely specified Boolean functions based on the representation of Rademacher - Walsh spectrum presented. The properties of such decompositions make them very suitable for the design with FPGAs [15]. A major advantage of the presented approach to Walsh spectrum calculation is its convenience for computer implementation and its ability to yield solutions to problems of very high dimensions.

6. REFERENCES

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

[2] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proc. of IEEE*, vol. 78, no. 2, pp. 264-300, Feb. 1990.

[3] M. J. Ciesielski, S. Yang, and M. A. Perkowski, "Minimization of multiple-valued logic based on graph coloring," *Technical Report, TR-CSE-90-13*, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, 1990. September 1990. Earlier version of this paper appeared as: "Multiple-valued minimization based on graph coloring," *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 262-265, 1989.

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

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

[6] B. J. Falkowski, and M. A. Perkowski, "Algorithms for the calculation of Hadamard-Walsh spectrum for completely and incompletely specified Boolean functions," *Proc. of 9th IEEE Int. Phoenix Conf. on Computers & Communications*, Scottsdale, AR, pp. 868-869, March 1990.

[7] B. J. Falkowski, and M. A. Perkowski, "Essential relations between classical and spectral approaches to analysis, synthesis, and testing of completely and incompletely specified Boolean functions," *Proc of 23rd IEEE Int. Symp. on Circuits & Systems*, New Orleans, LA, pp. 1656-1659, May 1990.

[8] 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," *Proc. of 33rd Midwest Symp. on Circuits & Systems*, Calgary, Alberta, pp. 1119-1122, August 1990.

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

[10] H. Landman, "Logic synthesis at Sun," *IEEE Conference Paper*, CH 2686-4/89/0000/0469, 1989.

[11] 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. Engng.*, vol. 5, pp. 231-249, 1978.

[12] 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," *Proc. of 22nd IEEE Int. Symp. on Circuits & Systems*, pp. 748-751, 1989.

[13] E. A. Trachtenberg, and D. Varma, "A design automation system for spectral logic synthesis," *Proc. of Int. Workshop on Logic Synthesis*, Research Triangle Park, NC, May 12-15, 1987.

[14] E. A. Trachtenberg, "Designing standard computer components using spectral techniques," *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 630-633, 1987.

[15] D. Varma, and E. A. Trachtenberg, "Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Trans. Computer-Aided Design*, vol. CAD-8, pp. 901-916, Aug. 1989.

[16] D. Varma, and E. A. Trachtenberg, "On the estimation of logic complexity for design automation applications," *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 368-371, Cambridge, MA, September 17-19, 1990.

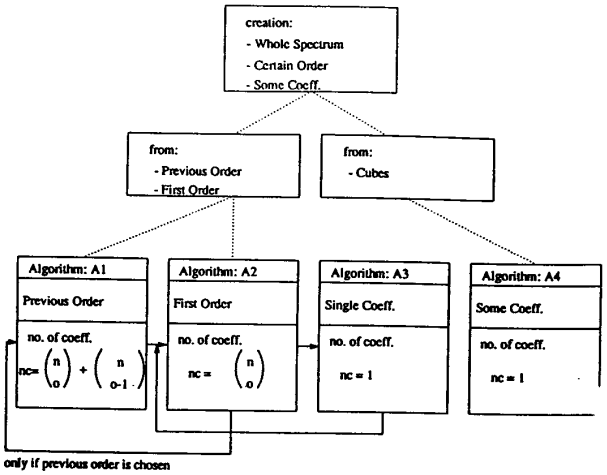

Fig. 1. Mutual relationships among the algorithms, the possible transitions and required memory sizes for each algorithm.

$x_1x_2x_3x_4$	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

Table 1.

SYMMETRY 27			
literals	type	previous	first
16	XX	6.3u 0.7s	9.9u 1.8s
16	1X	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	1X	35.1u 3.8s	106.1u 4.5s
20	XX	111.5u 21.3s	148.2u 32.4s
20	1X	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

Table 2. Execution times.