Computation of Spectral Information from Logic Netlists*

Rolf Drechsler

Albert-Ludwigs-University
Freiburg, Germany
drechsle@informatik.uni-freiburg.de

Mitch Thornton

Mississippi State University
Mississippi State, Mississippi
mitch@ece.msstate.edu

Abstract

Spectral information can be used for many CAD system tasks including synthesis, verification and test vector generation. We analyze the problem of extracting spectral information from Boolean and multi-valued logic netlists. It is shown that spectral information may be calculated directly from output probabilities and a method for extracting output probabilities from general graphs is described. As a special case, we consider AND/OR graphs which are a data structure recently proposed as an alternative to decision diagrams. Experimental results are given to demonstrate the efficiency of our approach.

1. Introduction

Spectral methods have been suggested as an efficient tool for circuit design in the early 1960s. Recently, they are of growing interest, since methods have been developed that allow for efficient solutions to logic synthesis problems [15, 9]. Spectral transformations are based on a linear transformation of the entire function table in the field of integers. The truth table vector, which has 2^n elements for binary valued functions where n denotes the number of variables of the function under consideration, is clearly exponential in size. For real circuits, spectral computation via the mathematical definition is impractical. Thus, more compact data structures are typically used for spectral calculations. Although many representations have been used in the past, each type has its' own advantages and disadvantages. The most popular data structure is the *Decison Diagram* (DD) and especially *Binary DDs* (BDDs) [3] have been successfully used in many applications. However, BDDs often become too large and for some functions, have an exponential size independent of the variable ordering [2]. Recently, the AND/OR decision graph has been proposed as an alternative for representing functions in CAD tools [20]. An overview of the use of this graph in the logic design arena is given in [7]. The AND/OR graph is attractive since in some cases it can provide a more compact representation than BDDs and it also allows for representing relationships whose detailed functionality is not yet known. However, for this particular type of data structure to be useful, it is imperative that methods are formulated that allow relevant information to be easily extracted by the CAD system.

Although the use of spectral methods in discrete logic dates back several years, there has been a recent renewal in the interest of these techniques largely due to new methods for the efficient computation of the spectra [4, 22]. The spectrum of a logic function imparts data concerning functionality and other characteristics.

In this paper, we show the close relation between various spectra and the output probabilities of a circuit. We propose a method for determining spectral information from a logic netlist and specifically, AND/OR graphs and describe extensions for using this method with *Multi-valued Decision Diagrams* (MDDs). In the next section, we introduce some preliminaries. Then we show how the computation of output probabilities is sufficient for determining several different types of spectra since the probability values and the spectral coefficients are related by simple algebraic expressions. Next, we formulate a technique to approximate the probabilities from a general graph representation that includes the AND/OR case as a subset. Finally, we present some experimental results followed by conclusions.

2. Preliminaries

A brief definition of the basic data structures and concepts will be given. The new methods described in this paper heavily depend on the use of these concepts.

2.1. Decision diagrams

As well-known, each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD), i.e. a directed acyclic graph where a Shannon decomposition

$$f = \overline{x}_i f_{x_i=0} + x_i f_{x_i=1} \quad (1 \leq i \leq n)$$

is carried out in each node.

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it contains neither isomorphic sub-graphs nor vertices with both edges pointing to the same node. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs. BDDs are a canonical representation, i.e. for each Boolean function the BDD can be uniquely determined. Furthermore, for functions represented by BDDs efficient manipulations are possible [3].

*This work was supported by DAAD grant 315/PPP/gü-ab and NSF grants CCR-9633085, SBE-9815371.

The extension to the MVL case is straightforward and the corresponding graphs are called *Multi-valued Decision Diagrams* (MDDs) [19].

2.2. Circuits and AND/OR graphs

In general, a *Combinational Logic Circuit* (CLC) is defined over a fixed library and modeled as a directed acyclic graph $C = (V, E)$ with some additional properties, typically in the form of graph vertex annotations. Very often a *standard library* (STD) consisting of primary input (PI) and output ports (PO), 2-input, 1-output gates (like NAND and NOR) and the 1-input, 1-output inverter *NOT* is used. If only AND and OR are allowed, a representation that maps 1-to-1 to AND/OR graphs is obtained. For more details see [7].

3. Relation of spectra and probability

This section will describe how output probabilities can be used to compute various spectral coefficients directly. The key idea here is to view the computation in terms of a vector-matrix product using an appropriate transformation matrix. Although such products are not computed directly, with this viewpoint each transformation matrix row vector can be considered as an output vector of some other function called a *constituent function*. To illustrate the generality of this approach, specific examples of global (Walsh), local (Reed-Muller) and Multi-resolution (Modified Haar) transforms are given.

By using Boolean relations between the function to be transformed, f, and the various constituent functions, f_c, output probabilities may be computed and used to calculate various spectral coefficients through simple algebraic relations. The advantage of this approach is that the spectral coefficients are computed individually allowing complete storage of 2^n spectral coefficients to be avoided. Furthermore, if efficient methods are used to compute the probability values such as those given in [13, 14, 22] a savings in computation time also results.

3.1. Walsh spectrum

The Walsh spectral values form several different spectra. The chief difference in the various spectra depend only upon the order that coefficients appear [10]. For example, this set of coefficients may be ordered naturally, in sequency order, in dyadic order, or, in revised sequency order. These orders yield the Hadamard-Walsh, the Walsh, the Paley-Walsh, and, the Rademacher-Walsh spectra, respectively [1, 10]. In the past, the Hadamard-Walsh has seen much use due to the fact that efficient decompositions of the transform matrix may be obtained allowing for 'fast transform' methods [5, 18] to be applied.

Since the relationships described here apply only to a single Walsh coefficient, the particular coefficient orderings are not important. Each coefficient is described by the particular f_c that corresponds to a transformation matrix row vector regardless of the actual location of the row vector in the matrix. The Walsh coefficient is denoted by $W_f(f_c)$ where f denotes that the spectral coefficient is with respect to function f, and, f_c denotes that the coefficient is dependent on the transformation matrix row vector corresponding to constituent function f_c.

It is generally the case that the Walsh spectral values are computed by replacing all occurrences of logic-1 with the integer, -1, and all occurrences of logic-0 with the integer, +1. The actual calculation of the coefficient is then carried out by using integer arithmetic. In terms of computing an inner-product of a transformation row vector and an output vector of a function to be transformed, it is easy to see that each "sub-product" term to be accumulated is either +1 or -1 in value. Furthermore, such a product of -1 will only occur when the functions f and f_c have different output values for the same set of input variable assignments.

If we consider the equivalence function given by the exclusive-NOR operation (XNOR), a function can be formed whose output is logic-1 if and only if both f and f_c are at logic-1, $\overline{f \oplus f_c}$. Furthermore, if the output probability of this function is computed, we have the percentage of outputs where both f and f_c simultaneously output the same value, $\wp\{\overline{f \oplus f_c}\}$. The corresponding Walsh spectral coefficient can then be obtained by scaling the output probability value by 2^n, where n is the number of variables in f. This is given in Equation 1.

$$W_f(f_c) = 2^n[1 - \wp\{f \oplus f_c\}] \tag{1}$$

The Walsh coefficients in Equation 1 depend upon the equivalence relation of the function being transformed and the constituent function. However, the spectral coefficient can be computed using Boolean operators other than XOR. This is accomplished by exploiting the probability relationships given in Equations 2 and 3.

$$\wp\{f + f_c\} = \wp\{f \cdot f_c\} + \wp\{f \oplus f_c\} \tag{2}$$

$$\wp\{f + f_c\} = \wp\{f\} + \wp\{f_c\} - \wp\{f \cdot f_c\} \tag{3}$$

Substituting the relationships in Equations 2 and 3 into Equation 1 yields the following results.

$$W_f(f_c) = 2^n[1 + 2(\wp\{f \cdot f_c\} - \wp\{f + f_c\})] \tag{4}$$

$$W_f(f_c) = 2^n[1 + 4\wp\{f \cdot f_c\} - 2\wp\{f\} - 2\wp\{f_c\}] \tag{5}$$

$$W_f(f_c) = 2^n[1 - 4\wp\{f + f_c\} + 2\wp\{f\} + 2\wp\{f_c\}] \tag{6}$$

The relationships in Equations 5 and 6 may be simplified since $\wp\{f_c\} = \frac{1}{2}$ is easily shown to be true for all f_c except the 0th ordered coefficient, $W_f(0)$, where $\wp\{0\} = 0$. This is because the rows of the Walsh transformation matrix can be viewed as output vectors of parity functions, thus, there are an equal number of logic-1 and logic-0 values. Also, the higher ordered Walsh coefficients can be related to the the 0th ordered coefficient through the expressions given in Equations 7, 8, and 9.

$$W_f(0) = 2^n(1 - 2\wp\{f\}) \tag{7}$$

$$W_f(f_c) = 2^n(4\wp\{f \cdot f_c\} - 1) + W_f(0) \tag{8}$$

$$W_f(f_c) = 2^n(3 - 4\wp\{f + f_c\}) - W_f(0) \tag{9}$$

3.2. Reed-Muller spectrum

The Reed-Muller family of spectra consist of 2^n distinct transformations. These are generally classified according to a *polarity number*. The polarity number is used to indicate if a literal is present in complemented or non-complemented form in the generalized Reed-Muller Boolean algebraic expression. In terms of the Reed-Muller spectra, the polarity number can be considered to uniquely define the transformation matrix. The generalized Reed-Muller spectra have been studied and used extensively in the past. References [6] and [8] provide detailed background material.

In relating output probabilities to the Reed-Muller spectra, considerations must be made due to the fact that the RM spectrum is computed over Galois field 2 and the output probabilities are real quantities in the interval [0,1]. An isomorphic relation is used to address this problem [21]. Like the Walsh spectrum, each of the rows of the RM transformation matrix may be viewed as constituent function output vectors. The constituent functions turn out to be all possible products of literals (with complementation or lack thereof) determined by the polarity number. Therefore, any arbitrary RM spectral coefficient may be computed as given in Equation 10.

$$R_f(f_c) = [2^n \wp\{f \cdot f_c\}](\mathrm{mod}2) \tag{10}$$

Using the probability relationships given in Equations 2 and 3, alternative relationships can be derived as given in the following.

$$R_f(f_c) = [2^n(\wp\{f\} - \wp\{f_c\} - \wp\{f + g\})](\mathrm{mod}2) \tag{11}$$

$$R_f(f_c) = [2^n(\wp\{f \oplus f_c\} - \wp\{f + g\})](\mathrm{mod}2) \tag{12}$$

3.3. Haar spectrum

Like the Walsh family of transforms, the output vector of the function to be transformed generally contains integers with -1 representing logic-1 and +1 representing logic-0. With this viewpoint, we can define the number of matches between a particular transformation matrix row vector as the number of times the row vector and function vector components are simultaneously equal to -1 or +1. This particular transformation matrix is referred to as the modified Haar transform since the non-zero entries in the transformation matrix are all normalized to ± 1 rather than having the values originally given in the Haar basis functions [10]. Note that the Haar transformation matrix contains a 0 value in addition to the 1 and -1 quantities which represent logic levels. Since some of the rows represent constituent functions that are cofactors, the output space is less than 2^3 and the presence of a 0 value acts as a place holder.

By applying the same principles as those used to determine the algebraic relationships between output probabilities and the Walsh family and RM transforms, we can formulate the Haar transform relationships. The presence of cofactors in the Haar constituent functions can be handled by using the conditional probability relationship to represent these quantities as output probabilities of the AND of the function to be transformed with its respective dependent literals. Note also that the maximum absolute value of a Haar spectral coefficient varies depending on the order of the coefficient. This is due to the reduction in the size of the range of the constituent functions containing cofactors. Equation 13 expresses the relationship between a particular Haar spectral coefficient, H_i, and probability expressions that evaluate whether the function to be transformed and the constituent function simultaneously evaluate to logic-0 (denoted as p_{m0}), or evaluate to logic-1 (denoted as p_{m1}).

$$H_k = 2^{n-i}[2(p_{m0} + p_{m1}) - 1] \tag{13}$$

4. Probability extraction from graphs

Techniques for the extraction of output probability values from generalized graphs are given here. Initially, we formulate a solution to an easy subset of problems and show that an exact value can be obtained. Next, the problem is generalized to cover any arbitrary form of graph (or netlist) including the AND/OR type and an approximate method is described.

4.1. Probability algorithm

To simplify the problem, the following assumptions are made:

1. Each leaf vertex in the AND/OR graph represents a unique variable in the support set of f, the Boolean function represented by the graph.

2. The AND/OR graph is a complete representation of the function f.

The restriction that each variable in the support set of f appears only once in a leaf vertex is equivalent to representing the function in a *Completely Fan-Out Free* (CFOF) tree-like circuit composed only of alternating levels of AND and OR gates. CFOF type circuits are much easier to use for spectral computations since no reconvergent fanout is present and all occurrences of the AND operation are then equivalent to multiplications of probability values (i.e. problems due to idempotence are avoided). $\wp\{f\}$ may be computed through a single traversal of the graph with CPU time requirements of $O(N)$ where N is the total number of vertices in the graph. In terms of spatial complexity, one additional word of storage per node is required for each vertex which will contain a value corresponding to the output probability of the subcircuit specified by assuming the particular vertex is a root. The algorithm is described by the following steps:

1. Assign each leaf node representing a unique x_i the value $\wp\{x_i\}$.

2. In a breadth-first fashion, traverse the graph from the leaf nodes toward the root.

3. As each vertex is visited, assign a probability value computed from all immediate children probability values and the appropriate rule based on the node's Boolean functionality.

4. $\wp\{f\}$ is given as the node probability of the graph root vertex.

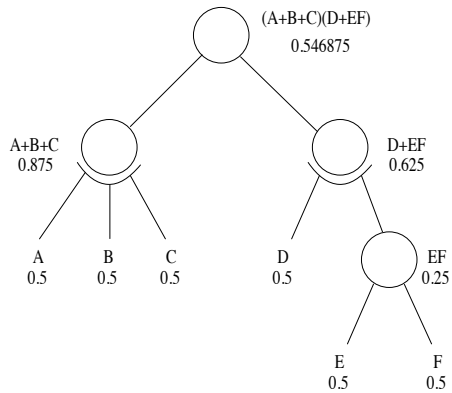

Figure 1. AND/OR Graph and Output Probabilities

In the case of AND/OR graphs, only two Boolean functionality characteristics apply; those for the AND and the OR operation. For the case where there are exactly two children vertices, these are given in Equations 14 and 15 respectively. For the general case where more than two children are present, the vertex can be considered as a "tree" of the two-child case.

$$\wp\{f_1 \cdot f_2\} = \wp\{f_1\} \cdot \wp\{f_2\} \tag{14}$$

$$\wp\{f_1 + f_2\} = \wp\{f_1\} + \wp\{f_2\} - \wp\{f_1\} \cdot \wp\{f_2\} \tag{15}$$

As an example of the application of this technique, consider the AND/OR graph in Figure 1. Each vertex is labeled with a Boolean expression representing the function represented by the corresponding subtree. Also, each vertex is labeled with a probability value that is actually the output probability of the subtree.

4.2. General graphs

In general, a vertex may assume any logic function. Thus, an arbitrary graph (not just AND/OR or max/min) may be utilized. If the same restrictions are applied to the general case, the only change in the algorithm is the type of rule applied based on the vertex's logical operator characteristic.

Two paths in a circuit are reconvergent if they begin at a common node (node A) and reconverge at another common node (node B). The paths are said to "fanout" at node A and reconverge at node B. Circuits which do not contain this reconvergent fanout are called tree circuits, and their output probabilities can be calculated in linear time by the algorithm given in [11]. If the primary input signals at each gate are independent as they are in tree circuits, then the output probability at each gate can be defined in terms of the probabilities of its inputs alone. Using the relationships described previously, the circuit output probability can be calculated by traversing the circuit from inputs to outputs and computing the probabilities at each intermediate gate. This calculation is linear with respect to the number of gates in the circuit since only a simple traversal of the circuit is needed. Using the relationships derived in the previous section, these exact output probabilities may then be used to compute spectral coefficients directly.

Allowing variables in the support set to appear in more than 1 leaf node is equivalent to the representation of a multi-level AND/OR circuit with reconvergent fanout. The computation of output probabilities for such representations was considered in the work by [16] and [17]. In [16] it was shown that the computation of the exact value of $\wp\{f\}$ using a netlist with reconvergent fanout requires the symbolic manipulation of a probability expression which, in the worst case, can have an exponential number of terms with respect to the number of support variables. The work in [17] proposed an approximation technique based on the notion of *supergates* [12]. This technique identifies the subgraph portions that contain the diverging and reconverging paths and replaces them with a single graph supernode or supergate.

Once the supergate for a node has been identified, the output probability for that node can be calculated, assuming that the output probability of each of the inputs to the supergate is known. Let X be the node for which the output probability is desired and the root of a supergate. Let $\underline{A}$ be a binary vector, $(a_1, a_2, \ldots, a_m)$, such that each bit in the vector is an assignment to one of the m fanout inputs of the supergate. The output probability of X given $\underline{A}$ can then be found by using the simple tree circuit relationships and traversing the supergate as if it had no reconvergent fanout. This process is repeated for the 2^m possible $\underline{A}$ vectors, and the output probability of X is given by the relationship in Equation 16.

$$P[X] = \sum_{\forall \underline{A}} P[X|\underline{A}] \cdot P[\underline{A}] \tag{16}$$

The value, $P[\underline{A}]$, is the probability of a given $\underline{A}$ vector and is found by taking the product of the probability of each bit in the vector. This calculation is obviously exponential with respect to the number of fanout inputs to the supergate, m. Once the output probabilities of each supergate have been computed in this fashion, the overall circuit output probability can be computed by invoking the tree circuit algorithm and treating each supergate vertex as an independent entity.

4.3. Supergate approximations

Although the method based on the use of supergates will yield an exact output probability and is practical for supergates with relatively few and relatively short reconvergent paths. Unfortunately, some netlists can contain supergates with a large number of fanout inputs (e.g. ISCAS85 benchmark, $c432$ contains supergates with as many as 35 fanout inputs). Our approach is to use the notion of a supergate coupled with a heuristic as given in [17] that approximates a supergate based on a "threshold" value that is related to the depth of the reconvergent fanout paths. This distance-based heuristic allows us utilize approximations of supergates resulting in reasonable computation time but with the tradeoff of an approximated overall output probability and ultimately, an approximated spectral coefficient value.

In this approach, each node is defined to be a distance, D, away from the node for which the supergate is to be determined. The distance from a node, N, to the supergate

node, X, is defined to be the length (number of nodes) of the shortest path from N to X (single input buffers and inverters are not counted). If the threshold value is a distance, T, then any node at a distance of T or greater is assumed to be an independent signal, and the supergate will terminate at no more than a distance of T from the supergate node.

The justification for this approach is that the farther a node is from a supergate node, the more it will"mix" with other signals and the less its effect will be on the supergate output. It is clear that if the threshold value is zero then all signals are considered independent and no approximate supergates will be formed. Alternatively, if the threshold is larger than the largest supergate, then the routine will calculate the output probability exactly since all supergates will be formed completely with none being approximated.

4.4. MVL logic networks

For logic networks with binary valued outputs and MVL inputs, all of the above derivations hold. This class of MVL networks are particularly important in logic synthesis since "characteristic equations" that represent multi-output Boolean functions are of this form.

It is also possible to formulate the case where MVL circuits contain MVL outputs, however the formulation of the probability based computations must be generalized in order to express the desired spectral coefficients properly.

5. Experimental results

An approximation method based on the use of *supergates* has been implemented and the results are compared to the true probability values. The true values were computed using a BDD representation of the benchmark circuits. This data is contained in Table 1 where the column labeled *Threshold* contains the depth of the logic levels used to compute a *supergate*. The *Average*, *Maximum* and *Percentage Error* values compare the approximated output probabilities using the *supergates* to the exact values based on BDD computations over all outputs in the benchmark circuits. The *CPU Time* column reports the time in seconds that was required for the computation of the probability of all outputs in the given netlist when this method was implemented and run on a 100 MHz Sun SPARCstation 20 with 160 MB of RAM using SunOS version 5.5.

The resulting output probability estimates can be used to directly compute corresponding spectral coefficients using the algebraic relationships presented in Section 3. Since the spectral coefficients are unique for each each circuit output, we have presented our experimental results in terms of output probability values. This allows us to give statistics over all of the circuit outputs rather than providing an huge table with individual spectral coefficients and their corresponding errors. In this table, we compute the average and maximum error as percentage differences from the true probability values obtained using an exact computation based on BDD representations with those obtained from the netlist formulation.

6. Conclusion

We can compute an exact $\wp\{f\}$ for a CFOF (tree-circuit) representation of f resulting in an exact value for the spectral coefficients. An approximation technique was implemented for the general case which includes AND/OR graphs as a subset. This technique was implemented and the relative error due to the approximation was computed. The results indicated that increasing the threshold level for the formation of the *supergates* did not significantly decrease the errors in some cases and is not a pragmatic improvement. It was also noted that the results were very mixed with very good approximations resulting for some benchmark functions.

The results indicate that this approach is indeed very effective for some benchmarks, however large errors result from others. The variance in error is not too large when compared to the overall error in terms of the threshold depth of the *supergate*. This indicates that the penalty in increased computation time for increased threshold values is not worthwhile. If the threshold value is allowed to continually increase, exact supergates will be formed at some point resulting in exact spectral coefficient computations. This fact coupled with the results of $c432$ and $c3540$ are interesting since increased threshold values (resulting in increased CPU runtimes) actually generated approximations with increased error in the output probabilities which would cause increased error in the corresponding spectral coefficients.

References

[1] P. W. Besslich. *Spectral Techniques and Fault Detection, (M. G. Karpovsky, editor)*. Academic Press, Boston, Massachusetts, 1985.

[2] R. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication. *IEEE Transactions on Computers*, vol. 40, no. 2:205–213, February 1991.

[3] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, vol. C-35, no. 8:677–691, August 1986.

[4] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral Transformations for Large Boolean Functions with Applications to Technology Mapping. *Proceedings of ACM/IEEE Design Automation Conference*, pages 54–60, 1993.

[5] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Computation*, vol. 19:297–301, 1965.

[6] M. Davio, J.-P. Deschamps, and A. Thayse. *Discrete and Switching Functions*. Mc-Graw-Hill, New York, New York, 1978.

[7] R. Drechsler, W. Kunz, D. Stoeffel, and A. Žužek. Decision diagrams and AND/OR graphs for design automation problems. *Proceedings of the International Conference on Information, Communication and Signal Processing*, pages 67–72, 1997.

[8] D. Green. *Modern Logic Design*. Addison-Wesley, Reading, Massachusetts, 1986.

Table 1. Results Using the Approximation Scheme for Probability Computation

Circuit	Inputs	Outputs	Threshold	Avg. Error	Max. Error	Pct. Error	CPU Time
c432	36	7	0	0.13167	0.35140	19.167	0.007
c432	36	7	1	0.13167	0.35140	19.167	0.007
c432	36	7	2	0.12203	0.32239	17.706	0.013
c432	36	7	3	0.13751	0.31966	20.042	2.497
c499	41	32	0	0	0	0	0.007
c499	41	32	1	0	0	0	0.007
c499	41	32	2	0	0	0	0.008
c499	41	32	3	0	0	0	0.020
c880	60	26	0	0.01915	0.05728	3.492	0.012
c880	60	26	1	0.01915	0.05728	3.492	0.013
c880	60	26	2	0.01867	0.05656	3.359	0.014
c880	60	26	3	0.00730	0.03311	1.360	0.025
c1355	41	32	0	0.00113	0.00113	0.2260	0.031
c1355	41	32	1	0.00113	0.00113	0.2260	0.032
c1355	41	32	2	0.00042	0.00079	0.0850	0.037
c1355	41	32	3	0	0	0	0.118
c1908	33	25	0	0.02994	0.20310	6.1020	0.034
c1908	33	25	1	0.02994	0.20310	6.1020	0.035
c1908	33	25	2	0.02990	0.20295	6.0940	0.162
c1908	33	25	3	0.01327	0.15656	2.1440	2.995
c3540	50	22	0	0.17646	0.33187	45.49	0.078
c3540	50	22	1	0.17646	0.33187	45.49	0.081
c3540	50	22	2	0.16712	0.33211	42.98	0.107
c3540	50	22	3	0.07353	0.18904	20.471	0.646

[9] J. P. Hansen and M. Sekine. Synthesis by spectral translation using boolean decision diagrams. *Design Automation Conference*, pages 248–253, June 1996.

[10] S. L. Hurst, D. M. Miller, and J. C. Muzio. *Spectral Techniques in Digital Logic*. Academic Press, Orlando, Florida, 1985.

[11] Savir J, G. Ditlow, and P. Bardell. Random pattern testability. *IEEE Transactions on Computers*, vol. C-33, no. 1:1041–1045, 1984.

[12] R. Krieger, B. Becker, and R. Sinković. A BDD-based algorithm for computation of exact fault detection probabilities. *Int'l Symp. on Fault-Tolerant Comp.*, pages 186–195, 1993.

[13] B. Krishnamurthy and I. G. Tollis. Improved Techniques for Estimating Signal Probabilities. *IEEE Transactions on Computers*, vol. 38, no. 7:1245–1251, July 1989.

[14] D. M. Miller. An improved method for computing a generalized spectral coefficient. *IEEE Trans. on CAD/ICAS*, vol. CAD-17, no. 3:233–238, March 1998.

[15] M. Miller. A spectral method for Boolean function matching. *European Design and Test Conference*, page 602, 1996.

[16] K. P. Parker and E. J. McCluskey. Probabilistic Treatment of General Combinational Networks. *IEEE Transactions on Computers*, vol. c-24:668–670, June 1975.

[17] S. C. Seth, L. Pan, and V. D. Agrawal. PREDICT-Probabilistic Estimation of Digital Circui t Testability. *Proceedings of the Fault Tolerant Computing Symposium*, pages 220–225, 1985.

[18] J. L. Shanks. Computation of the Fast Walsh-Fourier Transform. *IEEE Trans. Comp.*, vol. C-18:457–459, May 1969.

[19] A. Srinivasan, T. Kam, S. Malik, and R.E. Brayton. Algorithms for discrete function manipulation. In *Int'l Conf. on CAD*, pages 92–95, 1990.

[20] D. Stoffel, W. Kunz, and S. Gerber. AND/OR reasoning graphs for determining prime implicants in multi-level combinational networks. *ASP Design Automation Conference*, pages 529–538, 1997.

[21] M. A. Thornton and V. S. S. Nair. Fast Reed-Muller Sectrum Computation Uing Output Probabilities. *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 281–287, August 1995.

[22] M. A. Thornton and V. S. S. Nair. Efficient Calculation of Spectral Coefficients and Their Applications. *IEEE Trans. on CAD/ICAS*, vol. 14, no. 11:1328–1341, November 1995.