ELSEVIER

INTEGRATION, the VLSI journal 29 (2000) 101–116

INTEGRATION
theVLSI journal

www.elsevier.com/locate/vlsi

Boolean function representation and spectral characterization using AND/OR graphs☆

A. Žužek[a], R. Drechsler[b,*], M.A. Thornton[c]

[a]*Jožef Stefan Institute, Ljubljana, Slovenia*
[b]*Institute of Computer Science, Albert-Ludwigs-University, 79110, Freiburg im Breisgau, Germany*
[c]*Mississippi State University, Mississippi State, MS, USA*

Received 2 December 1999

Abstract

Methods based on AND/OR graph representations of Boolean relations provide a promising new way of approaching VLSI CAD design automation problems. AND/OR graphs can represent any Boolean network and they allow for systematic reasoning through the application of the technique of recursive learning. An approach to build and analyze AND/OR graphs that makes use of hashing techniques in a way similar to that for modern Decision Diagram (DD) packages is described. Additionally, the problem of extracting spectral information from AND/OR graphs is also examined. Spectral information can be used for many CAD system tasks including synthesis, verification and test vector generation. It is shown that spectral information may be calculated directly from output probabilities and a method for estimating output probabilities from AND/OR graphs is presented. Experimental results regarding the AND/OR graph package efficiency and the extraction of spectral information are provided.

1. Introduction

In order to solve a large class of problems in the area of VLSI CAD, an efficient underlying data structure is required. In the last few years several methods based on *decision diagrams* (DDs) have been proposed and used in industrial applications [1,2]. The most popular data structure in this area is the *Ordered Binary Decision Diagram* (OBDD) [3]. Typically, DD based methods allow this

☆This work was supported in part by NSF grants CCR-9633085, SBE-9815371 and DAAD grant 315/PPP/gü-ab.
* Corresponding author.
E-mail addresses: alenka.zuzek@ijs.si (A. Žužek), drechsle@informatik.uni-freiburg.de (R. Drechsler), mitch@ece.msstate.edu (M.A. Thornton).

0167-9260/00/$ - see front matter
PII: S0167-9260(00)00003-1

class of problems to be easily solved if the corresponding DD can be completely constructed. This is mainly due to the fact that a DD is a canonical representation of a corresponding Boolean function. The property of canonicity allows for the formulation of efficient DD manipulation algorithms.

An important problem in design automation is the determination of Boolean satisfiability. To solve this problem, it must be determined if a variable assignment exists such that a Boolean function can assume the logic value 1. Many problems in design automation as they occur in the context of formal verification, logic synthesis and VLSI testing can be mapped to the satisfiability problem. Since this problem can be solved efficiently if the DD has been constructed, DDs have become an integral part of many algorithms in VLSI CAD.

The major drawback of DDs is that they cannot represent all functions efficiently (i.e., the size of the DD grows exponentially with the number of variables of the Boolean function). In some cases, this characteristic holds for functions with high practical relevance such as multipliers [4].

When it is intractable to build a DD for the circuit under consideration, serious problems for DD based approaches arise. It is generally not possible to work with only partially constructed DDs since many methods require a completely specified representation. An efficient method for representing a Boolean function is mandatory for a CAD tool to be practically useful. Although many representations have been used in the past, each type has its own disadvantages.

Recently, the AND/OR decision graph has been proposed for representing functions in CAD tools. The use of AND/OR graphs as a new data structure in reasoning and logic synthesis was introduced in Ref. [5]. In Ref. [6] the main difference between the traditional approach based on DDs and the newer AND/OR approach is outlined. Since AND/OR graphs preserve the structural information of a given circuit, their size is proportional to a corresponding Boolean network. For some cases the network can also be too large to be easily handled. Therefore, it is necessary to represent AND/OR graphs in as compact a manner as possible. The need for reducing these graphs has been already pointed out in Ref. [7]. A package using AND/OR graphs as Boolean network representations has been developed that exploits the presence of isomorphism for reduction in required memory resources. Our experimental results indicate that sub-graph isomorphism frequently occurs in the AND/OR graph representations of logic networks. The method used for identifying and sharing the isomorphic sub-graphs using an open hashing technique is described.

While the AND/OR graph is a desirable representation since it can provide a more compact representation as compared to DDs, it is also beneficial in that it can allow for the representation of Boolean relationships whose detailed functionality is not yet known. However, for this characteristic to be useful, it is imperative that methods which allow for relevant information to be extracted from the graph are also be formulated. Here, we propose a method for determining spectral information from a AND/OR representation of a Boolean function. The use of spectral methods in digital 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. The spectrum of a logic function imparts data concerning functionality and other characteristics.

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. A technique to approximate the probabilities from a general graph representation that includes the AND/OR case as a subset is described.

The remainder of the paper is organized as follows. First, we discuss the types of symmetry present in AND/OR representations of logic functions and how this is used to reduce the overall storage requirements. The approach is validated by comparing the sizes of several benchmark circuit representations with and without the use of the isomorphism information. Next, we show how spectral information is algebraically related to the notion of output probabilities. Techniques for the exact and estimated output probabilities are described and results are given that show the relative error of the estimation technique versus the exact computation are provided. Finally, conclusions drawn from this work are given.

2. AND/OR graphs and the reasoning technique

AND/OR graphs can be used for a wide variety of problems and efficient techniques for traversing these graphs have been reported in the AI literature. In the area of VLSI CAD these graphs have been already successfully applied for algorithms for sequential error diagnosis [8,9]. But AND/OR graphs can also be used to represent Boolean functions since any Boolean expression can be understood as an AND/OR tree.

However such general AND/OR trees do not solve the important satisfiability problem. Just as DDs have to be constructed according to strict rules in order to have desired properties, it is also important to construct AND/OR trees in a specific way to make them useful for a given application. In Ref. [5] it has been shown that the nodes of an AND/OR graph can be associated with the various steps of a specific reasoning technique presented in Ref. [10] based on the notion of recursive learning. If AND/OR graphs are enumerated as described in Ref. [5] then they represent Boolean functions and fulfill two important properties,

- they solve the satisfiability problem and,
- they facilitate systematic reasoning in general multi-level Boolean networks (i.e. they allow us to identify implicants and implications in an easy way).

A detailed description of the application of AND/OR graphs in VLSI CAD methods is given in Ref. [7].

3. Reduction of AND/OR graphs

Enumeratio techniques that use AND/OR graphs for recursive learning have a tree-like structure. The space complexity of such graphs can become too large to be handled for some circuits (e.g. c432) providing motivation for representing the graphs in their most compact form. Sharing isomorphic sub-graphs in the AND/OR is one way to reduce the overall storage requirements.

In an AND/OR graph, AND and OR types of isomorphism can be found (i.e. isomorphic sub-graphs rooted in AND and isomorphic sub-graphs rooted in OR nodes). Both types of isomorphism are illustrated in Fig. 2 where the AND/OR graph represents the enumeration for ISCAS benchmark $c17$ given the initial assignment $1gat = 0$ ($c17$ is given in Fig. 1). Two pairs of isomorphic sub-graphs are denoted by the shaded areas. Sub-graphs rooted in $8 = 1$ are of an OR type. Both unjustified OR root nodes ($8 = 1$) have exactly the same two justification sets, $3gat = 0$

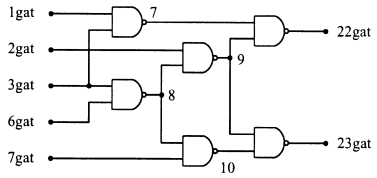


Fig. 1. ISCAS Benchmark circuit *c*17 [11].

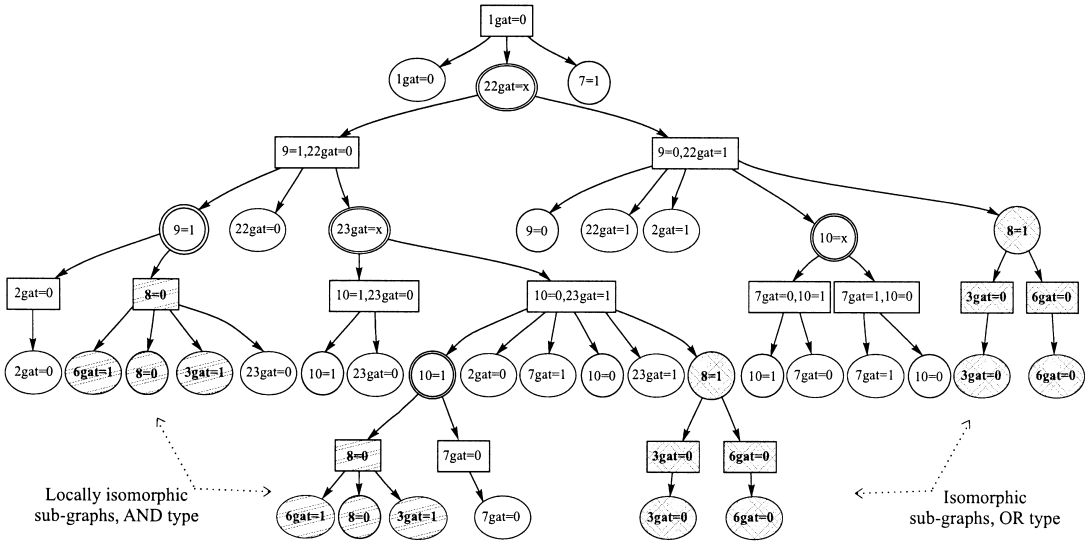


Fig. 2. Isomorphic sub-graphs of AND/OR graph for *c*17.

and $6gat = 0$. An AND type of isomorphism is also present in the same figure with regard to the two shaded sub-graphs are rooted in AND nodes labeled with $8 = 0$. It should be noted that this is a case of a *local isomorphism*, while the other case (OR type) depicts a *total isomorphism.* The lower sub-graph $22gat = x \rightarrow 23gat = x \rightarrow 10 = 1$ differs from the other in the set of OR nodes. It does not comprise two OR nodes labeled with $10 = 1$ and $23gat = 0$. These two signals in the circuit have already been assigned to the same values in the upper level of the AND/OR enumeration.

By analyzing various AND/OR graphs it can be observed that this locality of isomorphism often occurs. It can be present in the case of AND and OR types of isomorphism.[1]

A complete reduction of the AND/OR graph in Fig. 2 through the sharing of isomorphic sub-graphs and nodes is given in Fig. 3. The non-reduced AND/OR graph from the Fig. 2 comprises 51 AND and OR nodes whereas the reduced graph only has 37 nodes. In practice, reductions as large as 90% can be achieved (see Section 5).

[1] OR type of local isomorphism is absent in AND/OR graph for *c*17.

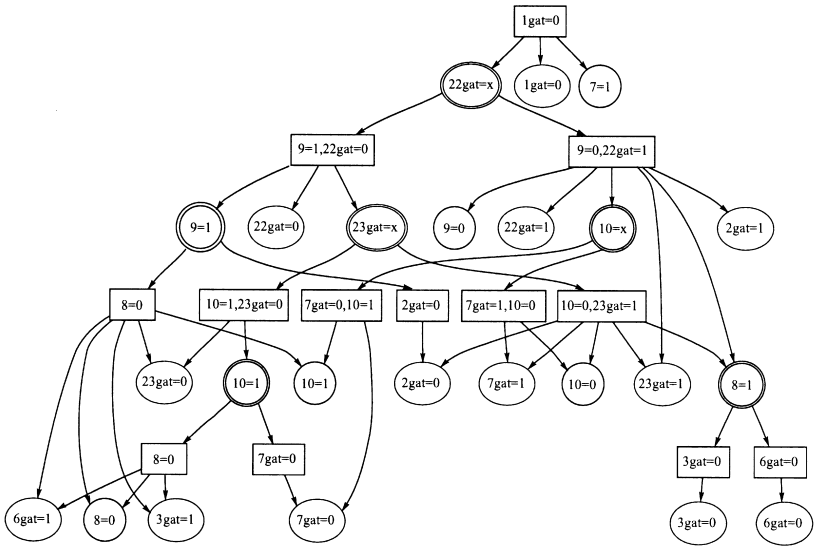


Fig. 3. Reduced AND/OR graph for $c17$.

4. Implementation

AND/OR graphs have alternating levels of AND and OR nodes, that is, each AND node has a successive set of OR nodes. In order to reduce the space complexity of the graph, AND nodes and their corresponding set of children OR nodes are combined into a single AND/OR node referred to as an AO-node. Such an implementation allows for the sharing of isomorphic sub-graphs rooted in AND nodes. In this way, sub-graphs rooted in OR nodes can be shared at lower levels of the graph.

The management of isomorphism in AND/OR graphs is similar to that of sharing nodes in BDDs [12,13]. The property of sharing isomorphic AND/OR graphs is preserved by using a hashing technique (i.e. recording all nodes in a hash table). Each time there is a need to create a new node, the hash table is accessed to determine if an instance of the new node is already present.

A difficulty arises in efficiently implementing shared, locally isomorphic, sub-graphs. For example, in the graph from Fig. 2, the sub-graph rooted in $8 = 0$ from the higher level (reached by $22gat = x \rightarrow 9 = 1$) is put into a hash table. When creating a sub-graph rooted in $8 = 0$, $22gat = x \rightarrow 23gat = x \rightarrow 10 = 1$, the AO-node with the same AND part but with the larger OR part is found. The difference between two nodes can be realized with the masking technique.

A 1 class **hsh_obj**

```
AOnode* obj;      /* pointer to the AOnode */
hsh_obj* next;    /* pointer to the next object */
```

4.1. Hash table for AND/OR graphs

The underlying structures used in the implementation of the hashing technique are given in A 1 and A 2. A strategy of *open hashing* is used where the hash table is represented as an array of lists of pointers to the AO-nodes in the graph. The AO-nodes that are addressed by the same list hash to the same hash-key value.

In order to produce an even distribution of hash-keys among the cells in the hash table, a size requirement of $3 \times$ the number of all nodes in the corresponding Boolean network results. Each third of the hash table covers one logic value (logic-0, logic-1 and don't care). A hash function is computed by decoding the values of AND part of the AO-node and its logic values. Each chain of nodes in the hash table has the same justification assignment node. Using such a function ensures good hash table performance.

A 2 class **hsh_table**

```
private:
hsh_obj          obj_ptr;
unsigned int     hsh_max;
AOnode*          look(AOnode* node, int flag);
public:
                 hsh_table(int size);
                 ~ hsh_table( );
void             clear_obj_list(int index);
AOnode*          locate(AOnode* node)
                        {return look(node, 0)};
void             insert(AOnode* node){look(node, 1)};
```

5. AND/OR graph reduction results

Experiments have been carried out using circuits from the LGSynth'93 benchmark set for the case of enumerating AND/OR graphs for all nodes of the network for both logic-0 and logic-1 Boolean values. These results were obtained using a 233 MHz PentiumII PC with 64 MB of main memory and 64 MB of swap space.

First, the presence of total and local isomorphism was identified. The resulting statistics on isomorphism presence are given in Table 1. The label *Bnet#* denotes the number of all nodes in the Boolean network and $\sum$ # is the sum of all AO-nodes. $\sum$ *tot-iso#* refers to the total isomorphism and shows the sum of all different AO-nodes. The next column, $\sum$ *loc-iso#*, gives the number of nodes in the AND/OR graph with sharing of locally isomorphic AO-nodes. *Max-totΔ* and *max-locΔ* represent the gain that is achieved by sharing total and local isomorphic sub-graphs. As can be seen, AND/OR graphs can have a lot of isomorphism, up to 60%. As expected, the results confirm that many sub-graphs are locally isomorphic (up to 90% for *con* 1).

Table 1
Experimental results of analyzing AND/OR graph for total and local isomorphism

Circuits	Bnet #	$\sum$ #	$\sum$ tot-iso #	$\sum$ loc-iso #	Max-totΔ (%)	Max-locΔ (%)
*c*17	11	346	317	254	20.00	48.00
*k*6	11	232	198	173	34.78	47.82
*k*35	11	212	194	181	30.77	40.00
Schneid	12	1265	943	366	42.42	79.78
Adder2	13	2142	1237	368	64.09	90.60
Con 1	18	4785	3203	799	56.64	91.48
*k*10	17	606	294	278	58.33	66.66

Table 2
Comparison of runtime and space complexity with the use of hash table vs. the use without any hashing

Circuits	Δspace	Δspace (%)	Time-hash	Time
*c*17	8560	39.6	0.06	0.05
*k*6	5192	45.3	0.03	0.02
*k*35	10328	31.4	0.13	0.12
Schneid	967	18.8	0.20	0.18
Adder2	18796	17.2	0.40	0.36
Con1	46068	18.0	1.09	1.02
*k*10	5144	51.8	0.03	0.02

The second part of the experiments compare run-time with space-complexity for AND/OR graph enumeration when using the hashing technique and when enumeration is performed without hashing. These results are presented in Table 2 and they depict the case of the total isomorphism. The first column, *Δspace*, gives the saved space in bytes, when using the hashing in comparison to running experiments without hashing. The results of relative space gain are provided in the next column. The last two columns measure the runtimes for both cases. The experiments running with hashing have slightly larger runtimes due to the fact that identification of isomorphic sub-graphs is performed after the sub-graph has been built. However, this slight increase in runtime overhead is negligible when compared to the dramatic savings in spatial requirements.

6. Relation of spectra and probability

This section describes 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–Müller) 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 Refs. [14–16] a savings in computation time also results.

6.1. 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 [17]. 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 [18,17]. 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 [19,20] 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 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 scalar product to be accumulated is either $+1$ or -1 in value. Furthermore, a scalar 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 Eq. (1).

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

The Walsh coefficients in Eq. (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 Eqs. (2) and (3).

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

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

Substituting the relationships in Eqs. (2) and (3) into Eq. (1) yields the following results:

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

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

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

The relationships in Eqs. (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 zeroth-ordered coefficient, $W_f(0)$, where $\wp\{0\} = 0$. Also, the higher-ordered Walsh coefficients can be related to the zeroth-ordered coefficient through the expressions given in Eqs. (7), (8), and (9).

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

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

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

6.2. Reed–Müller spectrum

The Reed–Müller 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–Müller Boolean algebraic expression. In terms of the Reed–Müller spectra, the polarity number can be considered to uniquely define the transformation matrix. The generalized Reed–Müller spectra have been studied and used extensively in the past. Refs. [21,22] provide in depth background material.

In relating output probabilities to the Reed–Müller 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 [23]. 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 Eq. (10).

$$R_f(f_c) = [2^n\wp\{ff_c\}] (\mathrm{mod}\, 2). \tag{10}$$

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

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

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

6.3. Haar spectrum

The details of the derivation for the Modified Haar transform are given in Ref. [24], an overview of the main results are given here. 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 $+1$. As an example of the modified Haar transform, consider the following transformation matrix for a 3-variable function is shown in Fig. 4. Each of the constituent Boolean functions are given to the left of their respective row vectors. Note that the 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. The following table contains symbols for each of the Haar coefficients, H_i, values that indicate the size of the constituent function range, i and n, 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}).

Using the notation introduced in Table 3, we can write an algebraic equation to compute the value of the kth Haar spectral coefficient in terms of the output probabilities used to compute p_{m0} and p_{m1}.

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

7. 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

$$\begin{array}{r} f \\ x_1 \\ x_2 \cdot f_{\overline{x}_1} \\ x_2 \cdot f_{x_1} \\ x_3 \cdot f_{\overline{x}_1\overline{x}_2} \\ x_3 \cdot f_{\overline{x}_1 x_2} \\ x_3 \cdot f_{x_1\overline{x}_2} \\ x_3 \cdot f_{x_1 x_2} \end{array} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Fig. 4. Example of modified Haar transformation matrix for $n = 3$.

Table 3
Relationship of the Haar spectrum and output probabilities

SYMBOL	i	$n-i$	p_{m1}	p_{m0}
H_0	0	3	$P[f\cdot 0]$	$P[\bar{f}\cdot\bar{0}]$
H_1	0	3	$P[fx_1]$	$P[\bar{f}\bar{x}_1]$
H_2	1	2	$\frac{P[f\bar{x}_1x_2]}{P[\bar{x}_1]}$	$\frac{P[\bar{f}\bar{x}_1\bar{x}_2]}{P[\bar{x}_1]}$
H_3	1	2	$\frac{P[fx_1x_2]}{P[x_1]}$	$\frac{P[\bar{f}x_1\bar{x}_2]}{P[x_1]}$
H_4	2	1	$\frac{P[f\bar{x}_1\bar{x}_2x_3]}{P[\bar{x}_1\bar{x}_2]}$	$\frac{P[\bar{f}\bar{x}_1\bar{x}_2\bar{x}_3]}{P[\bar{x}_1\bar{x}_2]}$
H_5	2	1	$\frac{P[f\bar{x}_1x_2x_3]}{P[\bar{x}_1x_2]}$	$\frac{P[\bar{f}\bar{x}_1x_2\bar{x}_3]}{P[\bar{x}_1x_2]}$
H_6	2	1	$\frac{P[fx_1\bar{x}_2x_3]}{P[x_1\bar{x}_2]}$	$\frac{P[\bar{f}x_1\bar{x}_2\bar{x}_3]}{P[x_1\bar{x}_2]}$
H_7	2	1	$\frac{P[fx_1x_2x_3]}{P[x_1x_2]}$	$\frac{P[\bar{f}x_1x_2\bar{x}_3]}{P[x_1x_2]}$

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.

7.1. 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. $\wp\{f\}$ 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.

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

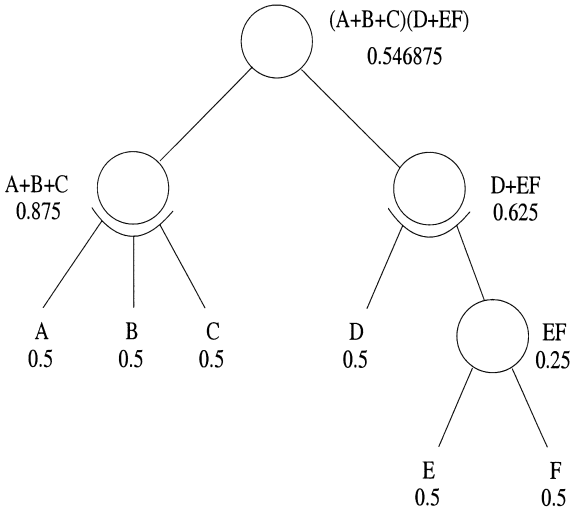


Fig. 5. AND/OR graph and output probabilities.

given in Eqs. (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.

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

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

As an example of the application of this technique, consider the AND/OR graph in Fig. 5. 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.

7.2. General graphs

In general a vertex may assume any Boolean function, thus, an arbitrary graph (not just AND/OR) 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 Boolean characteristic. Table 4 contains a list of all possible rules for all Boolean functions of two variables. In this table, the algebraic expression is given as a function of the two Boolean variables, a and b. The likelihood that the Boolean variables are a constant-1 value is denoted as $\wp\{a\} = A$ and $\wp\{b\} = B$.

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 re-convergent fanout. Such representations are a subset of the general case of any netlist represented as a connected graph with vertices corresponding to logic functions and edges to wires or nets. The computation of output probabilities for such representations was considered in the work by Ref. [25]. In Ref. [26] it was pointed out that the computation of the exact value of $\wp\{f\}$ using such representations can require the

Table 4
Probability rules for all possible Boolean characteristics of two variables

$f(a,b)$	$\wp\{f(a,b)\}$	$\wp\{f \mid A = B = \frac{1}{2}\}$
0	0	0
ab	AB	$\frac{1}{4}$
$a\bar{b}$	$A - AB$	$\frac{1}{4}$
a	A	$\frac{1}{2}$
$\bar{a}b$	$B - AB$	$\frac{1}{4}$
b	B	$\frac{1}{2}$
$a \oplus b$	$A + B - 2AB$	$\frac{1}{2}$
$a + b$	$A + B - AB$	$\frac{3}{4}$
$\overline{a + b}$	$1 - A - B + AB$	$\frac{1}{4}$
$\overline{a \oplus b}$	$1 - A - B + 2AB$	$\frac{1}{2}$
$\bar{b}$	$1 - B$	$\frac{1}{2}$
$a + \bar{b}$	$1 + B(A - 1)$	$\frac{3}{4}$
$\bar{a}$	$1 - A$	$\frac{1}{2}$
$\bar{a} + b$	$1 + A(B - 1)$	$\frac{3}{4}$
$\overline{ab}$	$1 - AB$	$\frac{3}{4}$
1	1	1

symbolic manipulation of a probability expression with an exponential number of terms with respect to the number of support variables. The work in Ref. [25] proposed an approximation technique based on the notion of *supergates*.

8. 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 5 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 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.

9. Conclusion

A method is introduced for reducing the size of AND/OR graphs when they are used as an underlying structure for representing Boolean networks. The experiments indicate sub-graph isomorphism occurs frequently in benchmark logic circuits; in some cases up to 90% of the graph

Table 5
Results using the approximation scheme for probability computation

Circuit	Inputs	Outputs	Threshold	Avg. error	Max. error	Pct. error	CPU time
*c*432	36	7	0	0.13167	0.35140	19.167	0.007
*c*432	36	7	1	0.13167	0.35140	19.167	0.007
*c*432	36	7	2	0.12203	0.32239	17.706	0.013
*c*432	36	7	3	0.13751	0.31966	20.042	2.497
*c*499	41	32	0	0	0	0	0.007
*c*499	41	32	1	0	0	0	0.007
*c*499	41	32	2	0	0	0	0.008
*c*499	41	32	3	0	0	0	0.020
*c*880	60	26	0	0.01915	0.05728	3.492	0.012
*c*880	60	26	1	0.01915	0.05728	3.492	0.013
*c*880	60	26	2	0.01867	0.05656	3.359	0.014
*c*880	60	26	3	0.00730	0.03311	1.360	0.025
*c*1355	41	32	0	0.00113	0.00113	0.2260	0.031
*c*1355	41	32	1	0.00113	0.00113	0.2260	0.032
*c*1355	41	32	2	0.00042	0.00079	0.0850	0.037
*c*1355	41	32	3	0	0	0	0.118
*c*1908	33	25	0	0.02994	0.20310	6.1020	0.034
*c*1908	33	25	1	0.02994	0.20310	6.1020	0.035
*c*1908	33	25	2	0.02990	0.20295	6.0940	0.162
*c*1908	33	25	3	0.01327	0.15656	2.1440	2.995
*c*3540	50	22	0	0.17646	0.33187	45.49	0.078
*c*3540	50	22	1	0.17646	0.33187	45.49	0.081
*c*3540	50	22	2	0.16712	0.33211	42.98	0.107
*c*3540	50	22	3	0.07353	0.18904	20.471	0.646

nodes are present in isomorphic sub-graphs. A method for identifying and sharing isomorphic sub-graphs was described and implemented using an open hashing technique. Experimental results indicate that this implementation resulted in significant savings in terms of required memory with a negligible increase in CPU run-time.

An exact $\wp\{f\}$ for a CFOF representation of f can be easily computed. 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 indicate that increasing the threshold level for the formation of the *supergates* did not significantly decrease the overall error and is not a pragmatic improvement. It was also noted that the results were mixed with very good approximations resulting for some benchmark functions.

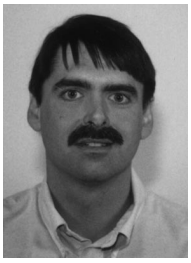
References

[1] S. Malik, A.R. Wang, R.K. Brayton, A. Sangiovanni-Vincentelli, Logic verification using binary decision diagrams in a logic synthesis environment, Proceedings of ICCAD, 1988, pp. 6–9.

[2] B. Becker, R. Drechsler, Decision diagrams in synthesis – algorithms, applications and extensions, VLSI Design Conference, 1997, pp. 46–50.
[3] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput. 8 (35) (1986) pp. 677–691.
[4] R.E. Bryant, On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication, IEEE Trans. Comput. 40 (1991) 205–213.
[5] D. Stoffel, W. Kunz, S. Gerbe, AND/OR reasoning graphs for determining prime implicants in multi-level combinational networks, Proceedings of ASP Design Automation Conference, 1997.
[6] R. Drechsler, W. Kunz, D. Stoffel, A. Žužek, Decision diagrams and AND/OR graphs for design automation problems, Proceedings of the International Conference on Information, Communication and Signal Processing, 1997, pp. 67–72.
[7] W. Kunz, D. Stoffel, Reasoning in Boolean Networks, Logic Synthesis and Verification Using Testing Techniques, Kluwer Academic Publishers, Boston, MA, 1997.
[8] A. Biasizzo, A. Žužek, F. Novak, Sequential diagnosis with asymmetrical tests, Comput. J. 41 (3) (1998) 163–170.
[9] R. Drechsler, A. Žužek, Efficient functional diagnosis for synchronous sequential circuits based on and/or graphs, Proceedings of the International Symposium on IC Technologies, Systems and Applications, 1997, pp. 312–315.
[10] W. Kunz, D.K. Pradhan, Recursive learning: a new implication technique for efficient solutions of CAD problems: test, verification and optimization, IEEE Trans. CAD 13 (9) (1994) 1143–1158.
[11] F. Brglez, H. Fujiwara, A Neutral Netlist of 10 Combinational Circuits and a Target Translator in FORTRAN, Proceedings of the International Conference on Circuits and Systems, 1985, pp. 663–698.
[12] S. Minato, N. Ishiura, S. Yajima, Shared binary decision diagram with attributed edges for efficient Boolean function manipulation, Proceedings of the ACM/IEEE Design Automation Conference, 1990, pp. 52–57.
[13] K.S. Brace, R.L Rudell, R.E. Bryant, Efficient implementation of a BDD package, Proceedings of Design Automation Conference, 1990, pp. 40–45.
[14] B. Krishnamurthy, I.G. Tollis, Improved techniques for estimating signal probabilities, IEEE Trans. Comput. 38 (7) (1989) 1245–1251.
[15] D.M. Miller, An improved method for computing a generalized spectral coefficient, IEEE Trans. CAD/ICAS CAD-17 (3) (1998) 233–238.
[16] M.A. Thornton, V.S.S. Nair, Efficient Calculation of Spectral Coefficients and Their Applications, IEEE Trans. CAD/ICAS 14 (11) (1995) 1328–1341.
[17] S.L. Hurst, D.M. Miller, J.C. Muzio, Spectral Techniques in Digital Logic, Academic Press, Orlando, FL, 1985.
[18] P.W. Besslich, M.G. Karpovsky (Eds.), Spectral Techniques and Fault Detection, Academic Press, Boston, MA, 1985.
[19] J.W. Cooley, J.W. Tukey, An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19 (1965) 297–301.
[20] J.L. Shanks, Computation of the fast Walsh–Fourier transform, IEEE Trans. Comput. C-18 (1969) 457–459.
[21] M. Davio, J.-P. Deschamps, A. Thayse, Discrete and Switching Functions, Mc-Graw-Hill, New York, 1978.
[22] D. Green, Modern Logic Design, Addison-Wesley, Reading, MA, 1986.
[23] M.A. Thornton, V.S.S. Nair, Fast Reed–Muller spectrum computation using output probabilities. Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed–Muller Expansion in Circuit Design, August 1995, pp. 281–287.
[24] M.A. Thornton, Modified Haar transform calculation using digital circuit output probabilities, Proceedings of the International Conference on Information, Communication and Signal Processing, 1997, pp. 52–58.
[25] S.C. Seth, L. Pan, V.D. Agrawal, PREDICT-probabilistic estimation of digital circuit *t* testability. Proceedings of the Fault Tolerant Computing Symposium, 1985, pp. 220–225.
[26] K.P. Parker, E.J. McCluskey, Probabilistic treatment of general combinational networks, IEEE Trans. Comput. c-24 (1975) 668–670.

Alenka Žužek received a B.Sc. degree in Electrical Engineering from the University Ljubljana, Slovenia, in 1993, and a M.Sc. degree in Computer Science from the same university in 1997. Currently she is working toward a Ph.D. degree in the field of VLSI CAD. She is a research assistant at Jožef Stefan Institute since 1994. Her research interests include AND/OR graph search algorithms for sequential diagnosis and the usage of binary decision diagrams and AND/OR graphs in the field of VLSI CAD.

Rolf Drechsler received his diploma and Ph.D. degree in Computer Science from the J.W. Goethe-University in Frankfurt am Main, Germany, in 1992 and 1995, respectively. He is currently working at the Institute of Computer Science at the Albert-Ludwigs-University of Freiburg im Breisgau, Germany. He is the Symposium's Chair of the IEEE International Symposium on Multiple-Valued Logic 1999 in Freiburg. He recently published two books at Kluwer Academic Publishers, one on BDD techniques co-authored by Bernd Becker and one on using evolutionary algorithms for VLSI CAD. His research interests include verification, logic synthesis, and evolutionary algorithms.



Mitch Thornton received the Ph.D. in Computer Engineering from Southern Methodist University in 1995. He is an Associate Professor in the Department of Electrical and Computer Engineering at Mississippi State University. From 1995 to 1999, he was a Fauclty member in the Department of Computer Systems Engineering at the University of Arkansas. Mitch also has over 5 years of experience as a design engineer at Raytheon E-Systems and Cyrix corporation. Research interests include algorithms for VLSI CAD, digital systems design and computer architecture.