IEEE European Design & and Test Conference
pages 438-443, Paris, March 1995

How many Decomposition Types do we need ? *

Bernd Becker Rolf Drechsler

Computer Science Department
Johann Wolfgang Goethe-University
60054 Frankfurt am Main, Germany
email: <name>@kea.informatik.uni-frankfurt.de

Abstract

Decision Diagrams (DDs) are used in many applications in CAD. Various types of DDs, e.g. BDDs, FDDs, KFDDs, differ by their decomposition types. In this paper we investigate the different decomposition types and prove that there are only three that really help to reduce the size of DDs.

1 Introduction

Decision Diagrams (DDs) are successfully applied in many fields of design automation, e.g. [17, 4, 1, 14, 7, 24, 11, 2, 9]. The most popular type of DD is the *Ordered Binary Decision Diagram* (OBDD) allowing efficient representation and manipulation of Boolean functions [5]. The more recent techniques have made it possible to handle (some) large functions without any basic variation of the OBDD concept itself. The dynamic variable ordering with sifting introduced by Rudell [21] allows to represent examples which could not be represented by any previous heuristic methods. Moreover, the variable ordering in [21] is handled by the package itself, alleviating the need for variable ordering before the actual processing. However, sifting tends to be very time consuming and the representation of large functions remains problematic; for certain classes of functions, notably multipliers, it has been known that OBDDs will be of exponential size irrespective of the order of the variables [6]. Thus, research has been geared towards variations of OBDDs. Among these variations there are those that utilize less restricted Decision Diagrams and there are other ones which augment BDDs with additional constructs. The constructs such as General BDDs [8], or pBDDs [12], IBDDs [13], XBDDs [14], and free BDDs [25] (also known as "1-time branching programs") remove the ordering constraint on BDDs at the expense of loosing the canonicity of the structure.

*This work was supported in part by DFG grant Be 1176/4-2

On the other hand there have been recent attempts at varying the nodes in BDDs. These include *Ordered Functional Decision Diagrams* (OFDDs) [16] and *Ordered Kronecker Functional Decision Diagrams* (OKFDDs) [10]. OFDDs and OKFDDs are canonical representations of Boolean functions. The nodes in OKFDDs are positive and negative Davio as well as Shannon nodes and thus are potentially more compact than OBDDs and OFDDs, both special kinds of OKFDDs [10]. Based on an equivalence decomposition *Ordered Equivalence Decision Diagrams* (OEDDs) [15] have been introduced. In [23, 22] further different types of DDs are presented.

Thus, a lot of new DD types have been (and are) introduced that use different decomposition types. In this paper we pose the important question:

> *How many decomposition types do exist and how many do we need for DDs? Do really all different decomposition types help to reduce the size of the correponding DD?*

In [3] it has been proven that there is an exponential trade-off between OBDDs and OFDDs for some Boolean functions. Thus, it seems reasonable to consider at least DDs that allow both decompositions - Shannon and Davio (positive and negative) - like OKFDDs.

In this paper we prove that these decompositions are the only ones that help to reduce the size of DDs if DDs with *Complemented Edges* (CEs) are considered. Other decompositions (like the equivalence decomposition [15]) are proven to be special cases of the ones already known.

The paper is structured as follows: In Section 2 various decomposition types are investigated. In Section 3 DDs are introduced and based on the decompositions of Section 2 GDDs are defined. In Section 4 CEs are introduced and the consequences for GDDs are discussed. We finish with a resume of the results in Section 5.

2 Decomposition Types

We investigate the different decompositions of a Boolean function f in two subfunctions so that it is possible to reconstruct f, if the subfunctions and the decomposition formula is known. For this we will first formally define a Boolean decomposition type.

Definition 1 A *decomposition type* $op \in \mathbf{B}_3$ is a function, such that for all $n \in \mathbf{N}$, for all $f \in \mathbf{B}_n$ and for all Boolean variables $x_i \in X_n := \{x_1, .., x_n\}$ it holds:

> There exist uniquely determined $g, h : \mathbf{B}^n \to \mathbf{B}$, such that $x_i \notin sup(g) \cup sup(h)$[1] and $f = op(x_i, g, h)$.[2]

Notice that the definition given above is more general than the approaches in [18, 20, 19], since we do not argue over *GF(2)*. One might expect that, because of this general definition, there exists a large number of decomposition types for Boolean functions. In the following we prove that this is not the case.

For $f \in \mathbf{B}_n$ define the *cofactor* f_i^0 of f with respect to $x_i = 0$ by $f_i^0(x) := f(x_1, .., x_{i-1}, 0, x_{i+1}, .., x_n)$ for $x = (x_1, x_2, .., x_n) \in \mathbf{B}^n$. Accordingly, f_i^1 denotes the *cofactor* f_i^0 of f with respect to $x_i = 1$. f_i^2 is defined as $f_i^2 := f_i^0 \oplus f_i^1$, $\oplus$ being the *Exclusive OR operation* and f_i^3 is defined as $f_i^3 := (f_i^0 \equiv f_i^1)$, $\equiv$ being the *negated Exclusive OR (or Equivalence) operation.*

Notation: Due to Definition 1 it is possible to write a decomposition type op as $op = (op_1, op_2)$ with $op_j : \mathbf{B}^2 \to \mathbf{B}$ $(j \in \{1, 2\})$ and

1. $op_1(g, h) := op(0, g, h) = f_i^0$
2. $op_2(g, h) := op(1, g, h) = f_i^1$

Theorem 1 Let $f : \mathbf{B}^n \to \mathbf{B}$ be a function over the set of variables X_n. Then there exist exactly 24 *different* decomposition types.

Proof: Since f_i^0 and f_i^1 can be any Boolean function, i.e. any combination of bits is possible for the outputs of f_i^0 and f_i^1, we may assume w.l.o.g. the existence of $x^{k,l} \in \mathbf{B}^n$ $(k, l \in \mathbf{B})$ such that

$$(f_i^0, f_i^1)(x^{k,l}) = (k, l).$$

We conclude that $(g, h)(x^{k,l}) \neq (g, h)(x^{k',l'})$ for $(k, l) \neq (k', l')$ $(k, l, k', l' \in \mathbf{B})$ as follows:

[1]$x_i \in sup(g)$, if and only if there exists $x = (x_1, x_2, .., x_n) \in \mathbf{B}^n$ with $g(x_1, .., x_{i-1}, 0, x_{i+1}, .., x_n) \neq g(x_1, .., x_{i-1}, 1, x_{i+1}, .., x_n)$.

[2]g (h) is called *left (right) decomposition function* of f.

Assume w.l.o.g $(g, h)(x^{0,0}) = (g, h)(x^{0,1})$. Then:

$$\begin{aligned}(0,0) &= (op_1(g,h)(x^{0,0}), op_2(g,h)(x^{0,0})) \\ &= (op_1(g,h)(x^{0,1}), op_2(g,h)(x^{0,1})) \\ &= (0,1)\end{aligned}$$

This means that the $x^{k,l}$ in combination with g and h compute all possible input combinations in $\mathbf{B}^2$.

It follows that the function table of op_i must consist of exactly two 0's and two 1's. (Otherwise all elements in $\mathbf{B}^2$ cannot be obtained as results of $(op_1, op_2)(g, h)(x^{k,l})$.) Thus the only functions possible for op_i are projections, inverse projections, exclusive-or and equivalence.

An analogous argumentation as above shows that it is not possible to only use exclusive-or and equivalence for both op_i's $(i \in \{1, 2\})$. Thus, one of op_i always has to be a projection or an inverse projection.

Let $proj_i$ $(\overline{proj}_i)$ be the (inverse) projection on the i-th component, *exor* be the exclusive-or and *equiv* the equivalence.

Then there are the following 24 different possible choices for $(op_1, op_2) \in DEC$ with

$$\begin{aligned}DEC = &\{proj_1, \overline{proj}_1\} \times \{proj_2, \overline{proj}_2\} \\ &\cup \{proj_2, \overline{proj}_2\} \times \{proj_1, \overline{proj}_1\} \\ &\cup \{proj_1, \overline{proj}_1, proj_2, \overline{proj}_2\} \times \{exor, \overline{equiv}\} \\ &\cup \{exor, equiv\} \times \{proj_1, \overline{proj}_1, proj_2, \overline{proj}_2\}\end{aligned}$$

The uniqueness of all decomposition functions is easy to prove. This proves the assertion of the theorem. □

As mentioned in the introduction we are looking for decompositions that really help to reduce the size of DDs. Obviously it makes no difference whether the projection $proj_1$ is used for op_1 and $proj_2$ for op_2 or $proj_2$ is used for op_1 and $proj_1$ for op_2. (It only leads to an exchange of the decomposition functions.)

Therefore we define:

Definition 2 Let $op = (op_1, op_2)$ and $op' = (op'_1, op'_2)$ be two decomposition types. op and op' are called *symmetric,* if $op_1 = op'_2$ and $op_2 = op'_1$.

Thus we can restrict the choice of (op_1, op_2) without loss of efficiency to the set DEC_{sym} with

$$\begin{aligned}DEC_{sym} = &\{proj_1, \overline{proj}_1\} \times \{proj_2, \overline{proj}_2\} \\ &\cup \{proj_1, \overline{proj}_1, proj_2, \overline{proj}_2\} \times \{exor, equiv\}\end{aligned}$$

We obtain:

Corollary 1 Let $f : \mathbf{B}^n \to \mathbf{B}$ be a function over the set of variables X_n. Then there exist exactly 12 *different* pairwise non-symmetric decomposition types.

name	op_1	op_2	g	h	*decomposition formula*
$D1$	$proj_1$	$proj_2$	f_i^0	f_i^1	$f = \overline{x}_i g + x_i h$
$D2$	$\overline{proj}_1$	$proj_2$	$\overline{f_i^0}$	f_i^1	$f = \overline{x_i + g} + x_i h$
$D3$	$proj_1$	$\overline{proj_2}$	f_i^0	$\overline{f_i^1}$	$f = \overline{x}_i g + \overline{\overline{x}_i + h}$
$D4$	$\overline{proj_1}$	$\overline{proj_2}$	$\overline{f_i^0}$	$\overline{f_i^1}$	$f = \overline{(x_i + g) \cdot (\overline{x}_i + h)}$
$D5$	$proj_1$	*exor*	f_i^0	f_i^2	$f = g \oplus x_i h$
$D6$	$\overline{proj}_1$	*exor*	$\overline{f_i^0}$	f_i^2	$f = \overline{g \oplus x_i h}$
$D7$	$proj_2$	*exor*	f_i^1	f_i^2	$f = g \oplus \overline{x}_i h$
$D8$	$\overline{proj_2}$	*exor*	$\overline{f_i^1}$	f_i^2	$f = \overline{g \oplus x_i h}$
$D9$	$proj_2$	*equiv*	f_i^1	f_i^3	$f = g \equiv (x_i + h)$
$D10$	$\overline{proj}_1$	*equiv*	$\overline{f_i^0}$	f_i^3	$f = \overline{g \equiv (\overline{x}_i + h)}$
$D11$	$proj_1$	*equiv*	f_i^0	f_i^3	$f = g \equiv (\overline{x}_i + h)$
$D12$	$\overline{proj_2}$	*equiv*	$\overline{f_i^1}$	f_i^3	$f = \overline{g \equiv (x_i + h)}$

Figure 1: Decomposition types

Before we investigate the remaining set in more detail we give a decomposition formula for each of the 12 decomposition types of the set in Figure 1. g (h) denotes the left (right) decomposition function. Notice, that the decomposition formula itself is not uniquely determined by the choice of $op = (op_1, op_2)$.

Decomposition type $D1$ is also known as the *Shannon expansion.* $D5$ and $D7$ are called *(positive and negative) Davio expansions* and $D9$ and $D11$ have been introduced in [15] as the *(positive and negative) Equivalence expansion.*

In the following we define a graph based representation of Boolean functions, called Decision Diagram, based on the decomposition types from Corollary 1. Using some Decision Diagram specific simplifications we prove that we do not really need all 12 decomposition types.

3 Decision Diagrams

In this section basic definitions of Decision Diagrams are presented.

Definition 3 A *Decision Diagram (DD)* over $X_n := \{x_1, x_2, .., x_n\}$ is a rooted directed acyclic graph $G = (V, E)$ with vertex set V containing two types of vertices, *non-terminal* and *terminal* vertices. A non-terminal vertex v is labeled with a variable from X_n, called the *decision variable* for v, and has exactly two successors denoted by $low(v), high(v) \in V$. All nodes with label x_i are denoted as *level* i. A terminal vertex v is labeled with a 0 or 1 and has no successors.

The size of a DD, denoted by $|DD|$, is given by its number of non-terminal nodes. If DDs are to be used as data structure in design automation, it turns out that a restriction to ordered DDs is convenient. A DD is *ordered*, if the variables are encountered at most once and in the same order on each path in the DD from the root to a terminal vertex. (In the following, letter O will be used to denote ordered DDs.)

Up to now DDs have not been related to Boolean functions. They can be related to Boolean functions by using the decomposition types from Corollary 1 and Figure 1.

Decomposition types are associated to the variables in X_n with the help of a *Decomposition Type List (DTL)* $d := (d_1, .., d_n)$ where $d_i \in \{D1, .., D12\}$. op_{Dj} ($j \in \{1, .., 12\}$) is used to denote the function in $\mathbf{B}_3$ corresponding to Dj.

Now, the General Decision Diagrams (GDD) can formally be defined as follows:

Definition 4 A *GDD over* X_n is given by a DD over X_n together with a fixed DTL, $d = (d_1, .., d_n)$. The function $f_G^d : \mathbf{B}^n \to \mathbf{B}$ represented by a GDD G over X_n with DTL d is defined as:

1. If G consists of a single node labeled with 0 (1), then G is a GDD for the constant 0 (1) function.
2. If G has a root v with label x_i and $d_i = D_j$, then G is a GDD for $f = op_{Dj}(x_i, f_{low(v)}, f_{high(v)})$, where $f_{low(v)}$ ($f_{high(v)}$) are the functions represented by the GDD rooted at $low(v)$ ($high(v)$).

Of course, a GDD is an OGDD iff the underlying DD is ordered.

A node in an GDD is called a *Dj-node* ($j \in \{1,..,12\}$) if decompostion type Dj from Figure 1 is applied at this node.

Due to the DTL, at every node of a fixed level i in the GDD, the same decomposition of type d_i is applied. This directly infers that GDDs are a generalization of BDDs, FDDs and KFDDs: If in all levels only Shannon decomposition ($D1$) is applied, the GDD will be a BDD. If only Davio decompositions ($D5$ and $D7$) are applied, the GDD will be an FDD. If only Davio and Shannon decompositions are used, but no Equivalence decompositions, the GDD will be a KFDD.

Reductions of different types are used to reduce the size of GDDs:

Type I: Delete a node v' whose label is identical to the label of a node v and whose successors are identical to the successors of v and redirect the edges pointing to v' to point to v.

Type Dj ($j \in \{1,..,12\}$)**:** Delete a node v if the function represented at this node is independent of the correponding variable. Connect the incoming edges of the deleted node to the corresponding successor.

While each node in a GDD is a candidate for the application of reduction type I, the type Dj may be different dependant on the decomposition type.

Example 1 For $D1$ and $D5$ the graphical interpretation is shown in Figure 2, i.e. nodes are deleted iff the variable is not in the support of the function being represented.

A GDD is *reduced* iff no reductions can be applied to the GDD. Two GDDs, G_1 and G_2, (with the same DTLs) are called *equivalent* iff G_2 results from G_1 by repeated applications of reductions and inverse reductions. A GDD, G_2, is called the *reduction* of a GDD, G_1, iff G_1 and G_2 are equivalent and G_2 itself is reduced.

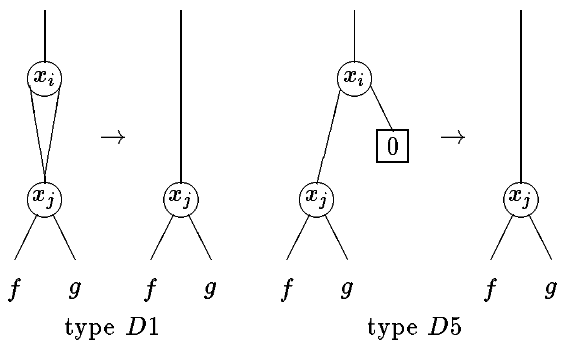

Figure 2: Reduction types for $D1$ and $D5$

From [10], it is known that reductions can be used to define canonical representations for not only OBDDs and OFDDs, but also for OKFDDs. The result directly transfers to OGDDs, if a fixed DTL is considered.

In the following we restrict to OGDDs but most of the results can be transfered to the general case.

4 Optimization of Graph-Size

In Section 3 it has been described how the size of GDDs can be decreased using reductions of various types. Since we are interested in small representations of Boolean functions we now present a method to further optimize the size of an OGDD. This method also has direct consequences for the choice of decomposition types for OGDDs.

In the following we indicate that *Complemented Edges* (CEs), that are used for other ODDs such as OBDDs [4] and OKFDDs [10], can also be used for OGDDs. CEs are used for the representation of a function and its complement by the same node. The constraints of CEs to maintain a canonical form for OKFDDs were given in [10]. The generalization to all other types is straighforward.

The difference between Dj types ($j \in \{1,..,12\}$) is that different restrictions have to be imposed on where CEs are set in order to obtain a canonical form. The functionally equivalent pairs are shown in Figure 3 for the case of type $D1$ and in Figure 4 for the case of type $D5$ where a dot on a line symbolizes a complement edge. Here always the left representative is used, i.e., the *low*-edge (f^0-edge) must be a regular - uncomplemented - edge. It then follows that reduced OGDDs with complemented edges are unique.

For the storage of the additional information one bit is needed. As suggested in [4] the overhead can

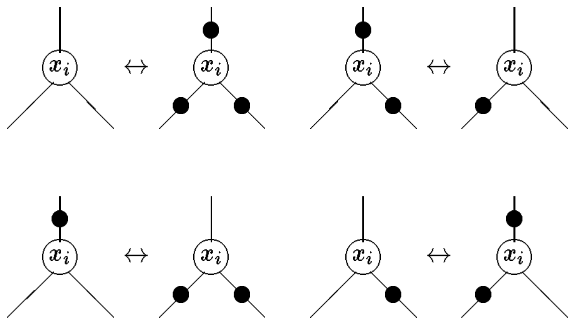

Figure 3: Identification for canonical form for $D1$

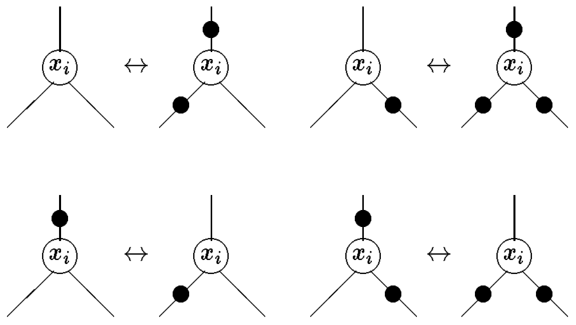

Figure 4: Identification for canonical form $D5$

be saved if the lowest bit of each pointer is used on machines where this is allowed.

If we now take a closer look at the *new* OGDDs, i.e., OGDDs with CEs. We can see that we do not need all decomposition types proposed in Section 2:

In the following we investigate the relation between decomposition types that result from each other, if the left and (or) right decomposition function is complemented. The following three sets defining a partition of DEC_{sym} can be distinguished

$$\begin{aligned} DEC_S &= \{D1, D2, D3, D4\} \\ DEC_{pD} &= \{D5, D6, D10, D11\} \\ Dec_{nD} &= \{D7, D8, D9, D12\}, \end{aligned}$$

DEC_S (DEC_{pD}, DEC_{nD}) being the set of decomposition types that contains the Shannon decomposition (positive Davio decomposition, negative Davio decomposition). We now prove that changing decomposition types within the sets DEC_S, DEC_{pD}, DEC_{nD} does not change the size and structure of an OGDD. Consider for example the *positive Davio* (D5) and the *negative Equivalence (D11) decomposition* of a function f (see Figure 1). The left decomposition functions are identical, the right decomposition functions are complements of each other. It follows that a decomposition type of a variable x_i can be changed from D11 to D5 by setting (or removing) a complement mark on the *high*-edges of level i. Analogously all other changes within the sets defined above can be realized. If the resulting OGDD is not canonical, the complement marks in the level $\leq i$ are adjusted according to the rules given by the functionally equivalent pairs (see e.g. Figures 3 and Figure 4). We conclude:

Lemma 1 Consider DTLs $d = (d_1, d_2, \ldots, d_n)$ and $d' = (d'_1, d'_2, \ldots, d'_n)$ with $\{d_i, d'_i\}$ either subset of DEC_S or DEC_{pD} or DEC_{nD} for $1 \leq i \leq n$. Let G be the (reduced) canonical OGDD with DTL d for the Boolean function f.

> Then the (reduced) OGDD G' with DTL d' has identical size and structure as G. The complement marks might be distributed differently.

Therefore, it does not make sense to consider more than one representative from the sets DEC_S, DEC_{pD}, DEC_{nD}, e.g. it is not necessary to consider both decompositions, the Davio and the Equivalence decomposition. Instead it is sufficient to restrict oneself w.l.o.g. to the Shannon, the positive and negative Davio decomposition. All in all, we obtain:

Theorem 2 For each OGDD G there exists an OKFDD G' of the same size and structure being a canonical representation of the function represented by G.

5 Conclusions

We investigated the number of different decomposition types for *Decision Digrams.* We proved that there exist only three relevant different unique decompositions of a Boolean function in two subfunctions.

Our result prevents from introducing more decomposition types than the ones that can really help to reduce the size.

One important observation is that OKFDDs as presented in [10] are the most general DD that can be considered for *ordered* DDs.

References

[1] P. Ashar, A. Ghosh, S. Devadas, and A.R. Newton. Combinational and sequential logic verification using general binary decision diagrams. In *Int'l Workshop on Logic Synth.*, 1991.

[2] B. Becker and R. Drechsler. Synthesis for testability: Circuits derived from ordered kronecker functional decision diagrams. In *European Conf. on Design Automation*, 1995.

[3] B. Becker, R. Drechsler, and R. Werchner. On the relation between bdds and fdds. In *LATIN95, LNCS*, 1995.

[4] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[5] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 8:677–691, 1986.

[6] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, 40:205–213, 1991.

[7] R.E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM, Comp. Surveys*, 24:293–318, 1992.

[8] J.R. Burch. Using BDDs to verify multipliers. In *Design Automation Conf.*, pages 408–412, 1991.

[9] R. Drechsler and B. Becker. Sym*pathy*: Fast exact minimization of fixed polarity reed-muller expressions for symmetric functions. In *European Conf. on Design Automation*, 1995.

[10] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams. In *Design Automation Conf.*, pages 415–419, 1994.

[11] R. Drechsler, M. Theobald, and B. Becker. Fast OFDD based minimization of fixed polarity reed-muller expressions. In *European Design Automation Conf.*, pages 2–7, 1994.

[12] S. J. Friedman. *Efficient Data Structures for Boolean Function Representation.* PhD thesis, Dept. of Comput. Sciences, Princeton University, 1990.

[13] J. Jain, M. Abadir, J. Bitner, D. Fussell, and J. Abraham. IBDDs: An efficient functional representation for digital circuits. In *European Conf. on Design Automation*, pages 441–446, 1992.

[14] S.-W. Jeong, B. Plessier, G. Hachtel, and F. Somenzi. Extended BDD's: Trading of canonicity for structure in verification algorithms. In *Int'l Conf. on CAD*, pages 464–467, 1991.

[15] U. Kebschull. Die Äquivalenz-Expansion Boolescher Funktionen. Technical report, WSI-93-15, Universität Tübingen, 1993.

[16] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *European Conf. on Design Automation*, pages 43–47, 1992.

[17] S. Malik, A.R. Wang, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Int'l Conf. on CAD*, pages 6–9, 1988.

[18] M.A. Perkowski. The generalized orthonormal expansion of functions with multiple-valued inputs and some of its application. In *Int'l Symp. on multi-valued Logic*, pages 442–450, 1992.

[19] M.A. Perkowski. A fundamental theorem for exor circuits. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, pages 52–60, 1993.

[20] M.A. Perkowski, A. Sarabi, and F. Beyl. XOR canonical forms of switching functions. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, pages 27–32, 1993.

[21] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[22] T. Sasao. EXMIN2: A simplification algorithm for Exclusive-OR-Sum-of products expressions for multiple-valued-input two-valued-output functions. *IEEE Trans. on CAD*, 12:621–632, 1993.

[23] T. Sasao. *Logic Synthesis and Optimization.* Kluwer Academic Publisher, 1993.

[24] C.C. Tsai and M. Marek-Sadowska. Boolean matching using generalized Reed-Muller forms. In *Design Automation Conf.*, pages 339–344, 1994.

[25] I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM*, 35(2):461–471, 1988.