Minimization of 2-level AND/XOR Expressions using Ordered Kronecker Functional Decision Diagrams *

Bernd Becker Rolf Drechsler Michael Theobald

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

Abstract

We present two new classes of 2-level AND/XOR expressions: the class RKRO of Reduced Kronecker Expressions and the class GKRO of Generalized Kronecker Expressions. GKRO contains RKRO and KRO, the well-known class of Kronecker Expressions [Sas93b] as a subclass. There is a close relation between RKROs, KROs, GKROs and Ordered Kronecker Functional Decision Diagrams (OKFDDs) [DST+94], that together with efficient OKFDD algorithms can be utilized for (exact and heuristical) minimization of RKROs, KROs and GKROs.

In this paper we concentrate on RKROs and KROs and propose several algorithms for their minimization. Experimental results are given to show the efficiency of our approach. For the first time efficient minimized 2-level AND/XOR expressions are determined for benchmark functions with more than 100 variables. Furthermore, we compare our solutions to results obtainable for other classes of AND/XOR forms. In particular, RKROs are much smaller than FPRMs and also turn out to be superior to KROs. In many cases they are even as compact as GKROs.

1 Introduction

The high complexity of modern VLSI circuitry has shown an increasing demand for synthesis tools. In the last few years synthesis based on AND/XOR realizations has gained more and more interest [FTY87, PCSS92]. This is due to several reasons: AND/XOR logic often leads to a more compact realization of switching functions than AND/OR, which is especially true for arithmetic circuits, error correcting circuits and circuits for tele-communication [Ber68, SB90, Sau92, Rol93]. Additionally, circuits with XOR gates have good testability properties [Red72, SR75, PCSS92, DB93, SP93, BD94b] - at least if they are restricted to specific subclasses of

*This work was supported in part by DFG, Grant Be 1176/4-1.

AND/XOR forms - and thus are well suited for design for testability. Last, but not least, new technologies make realizations with XOR gates more practical; the circuits can easily be mapped to field programable gate arrays (FPGAs) [BFRV92], where XOR gates, e.g. in table-lookup FPGAs, do not cause any extra cost in terms of chip area. Taken together, this calls for more integration of AND/XOR logic into mainstream synthesis.

Whereas powerful minimization tools for AND/OR based synthesis have been developed in the last few years [BHMS84, CFM93], we see a lack of adequate tools for minimization of AND/XOR circuits with a large number of inputs. There exist several differing classes of of (multi-level and 2-level) AND/XOR forms [Per92, Sas93b]. For this paper we concentrate on 2-level forms.

The most general 2-level form is the XOR sum-of-product expression (ESOP). Several tools for minimization have been presented. Most of the exact algorithms can only be used for functions with a small number of variables [Pap79, PC90, SP91, Sas93a]. For an overview and a comparison between the approaches which often try to apply methods known from an "ESPRESSO-like" AND/OR minimization see [Sas93b]. ESOPs are not canonical and the minimization problem turns out to be very hard to solve. Therefore, besides approaches to attack the problem in its general form, restricted versions of ESOPs have been considered. Additionally, the testability of the restricted forms is often better than the testability of general ESOPs. Several methods for minimization of Fixed-Polarity Reed-Muller Expressions (FPRMs), i.e. fixed polarity AND/XOR canonical representations, have been considered [SP91, SP92, TM93, DTB94]. For this class efficient algorithms are known also for large functions, but due to the strong restrictions imposed on the representations a "large number" of Boolean functions has no efficient FPRM.

Therefore, in this paper we consider two new, more general classes, i.e. superclasses of FPRM: the class RKRO of Reduced Kronecker Expressions and the class GKRO of Generalized Kronecker Expressions. It is shown that RKRO and also the well-known class of Kronecker Expressions (KROs) [Sas93b] are subclasses of GKRO. There is a close relation between RKROs, KROs, GKROs and a multi-level data structure, called Ordered Kronecker Functional Decision Diagrams (OKFDDs) [DST$^+$94]. RKRO and KRO as subclasses of GKRO result, if we restrict to consideration of reduced and complete OKFDDs, respectively. Synthesis operations on OKFDDs have been developed and integrated in an OKFDD package [DST$^+$94]. It turns out in this paper, that OKFDDs together with these efficient synthesis operations can be successfully applied for the minimization of RKROs, KROs and GKROs.

In this paper we concentrate on RKROs and KROs and propose several algorithms for their minimization. All algorithms proposed for RKROs can be directly translated to the "complete" case and thus result in minimization methods for KROs. We start with a method for exact minimization of RKROs and KROs for single and multiple output functions. The terms of the 2-level representation are computed by simple traversal of the corresponding OKFDD in the single output case. For multiple output functions we use a recursive algorithm on the OKFDD that is well-known as union operation for zero-suppressed Ordered Binary Decision Diagrams [Min93]. Our minimization algorithm is mainly limited by the running time, but not the space requirement. We succeeded in minimizing functions with up to 15 variables in acceptable time. For larger functions heuristics for minimization of RKROs and KROs are used to obtain good results. Experimental results are given to show the efficiency of our approach. We succeeded in heuristically minimizing functions with more than 100 variables. In all cases we obtained RKROs which are (significantly) smaller than the corresponding minimized KROs. In this sense RKROs are superior to KROs, although this is mathematically correct only for single output functions.

It is an interesting question, whether minimization in the larger class GKRO will result in still smaller 2-level representations. We give the following partial answer to this question: For single output functions the minimum GKRO and minimum RKRO are identical. Thus, we have implicitly computed the minimum GKRO, which, of course, is always smaller ($\leq$) than the minimum KRO. The situation changes, if we consider multiple output functions. We show by an example that theoretically the minimum GKRO and even the minimum KRO may be smaller ($<$) than the minimum RKRO. The canonical algorithm for the construction of a minimum GKRO for the general case is computationally very expensive. We therefore developed a heuristic, that guarantees the construction of a GKRO that is at least as small as the corresponding KRO. In almost all our experiments the resulting GKRO was indeed smaller than the corresponding KRO, but in general we did not obtain the quality of the RKROs. We therefore consider minimization of RKROs to be a good compromise between computational complexity and quality of the solution also for the class GKRO.

The paper is structured as follows: In Section 2 notations and definitions are given. OKFDDs, RKROs, KROs and GKROs are defined. The close relation between OKFDDs on the one hand and RKROs, KROs and GKROs on the other hand is shown. The problem of exactly minimizing RKROs and KROs of functions with several outputs is investigated in more detail. Algorithms for exact and heuristical minimization of RKROs, KROs and GKROs using OKFDDs are given in Section 3. In Section 4 experimental results are presented. A summary of the results is given in Section 5.

2 Preliminaries

In this section we define the basic data structure used in this paper and investigate its close relation to a special class of AND/XOR expressions.

2.1 Ordered Kronecker Functional Decision Diagrams

The core of the OKFDD data structure for Boolean functions is a decision diagram (DD), which is a directed acyclic graph with some additional properties.

Definition 1 A *decision diagram (DD)* over $X_n := \{x_1, x_2, \ldots, x_n\}$ 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 $low(v), high(v) \in V$. 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 internal nodes. If DDs are to be used as a data structure in design automation, it turns out that further restrictions on their structure will be necessary. Three such restrictions are defined below:

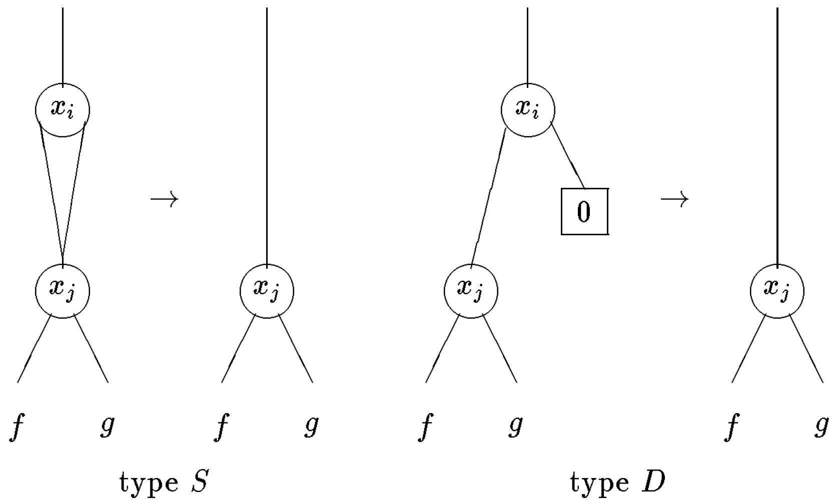

Figure 1: Reduction types

Definition 2 A DD is *complete* if each variable is encountered exactly once on each path in the DD from the root to a terminal vertex. A DD is *free* if each variable is encountered at most once on each path in the DD from the root to a terminal vertex. A DD is *ordered* if it is free and the variables are encountered 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 describe ordered DDs.

It is possible to define certain reductions on the decision diagrams in order to reduce their size. Three reduction types will be used.

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

Type S: Delete a node v whose two outgoing edges point to the same node and connect the incoming edges of the deleted node to the corresponding successor.

Type D: Delete all nodes v whose successor $high(v)$ points to the terminal 0 and connect the incoming edges of the deleted node to the corresponding successor.

In Figure 1, the graphical representation of reductions of type S and D are shown. While each node in a DD is a candidate for the application of reduction type I, candidates for the application of the remaining reduction types have to be defined in advance: The non-terminal nodes of a DD are partitioned in two classes, Shannon nodes and Davio nodes. Reduction of type S (D) is only allowed at Shannon (Davio) nodes. For the following assume that this partition is given.

Definition 3 A DD is *reduced* if no reductions can be applied to the DD. Two DDs, G_1 and G_2, are called *equivalent* iff G_2 results from G_1 by repeated applications of reductions and inverse reductions. A DD, G_2, is called the *reduction* of a DD, G_1, if G_1 and G_2 are equivalent and G_2 itself is reduced.

A careful analysis of the proofs in [Bry86, GM94, SW93] shows that the following lemma is valid for DDs:

Lemma 1 The reduction of an ordered DD, G, is uniquely determined provided that a partition of the variables in Shannon and Davio nodes is given. The reduction can be computed in linear time in the size of G.

Until now it has not been defined how DDs can be related to Boolean functions. To do this, the following notions are helpful. All nodes labeled with the same variable are denoted as a *level* in the following. Let $f : \mathbf{B}^n \to \mathbf{B}$ be a Boolean function over the variable set X_n. Then f_i^0 denotes the *cofactor* of f with respect to $x_i = 0$, defined by $f_i^0(x) := f(x_1, .., x_{i-1}, 0, x_{i+1}, .., x_n)$ for $x = (x_1, x_2, \dots, x_n) \in \mathbf{B}^n$. Similarly, f_i^1 denotes the cofactor for $x_i = 1$. Finally, f_i^2 is defined as $f_i^2 := f_i^0 \oplus f_i^1$. (Notice that the three functions f_i^0, f_i^1, f_i^2 can naturally be interpreted as Boolean functions from $\mathbf{B}^{n-1}$ to $\mathbf{B}$ defined over the variables $x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n$.) Using the above definitions, the following decompositions can be proven for an arbitrary Boolean function f:

$$f = \overline{x}_i f_i^0 + x_i f_i^1 \qquad \textit{Shannon decomposition} \tag{1}$$

$$f = f_i^0 \oplus x_i f_i^2 \qquad \textit{positive Davio decomposition} \tag{2}$$

$$f = f_i^1 \oplus \overline{x}_i f_i^2 \qquad \textit{negative Davio decomposition} \tag{3}$$

Now, the Ordered Kronecker Functional Decision Diagrams can formally be defined as follows:

Definition 4 An *OKFDD over X_n* is given by an ordered DD over X_n together with a uniquely determined decomposition type, $d_i \in$ {Shannon, positive Davio, negative Davio}, assigned to each variable x_i ($i \in \{1, .., n\}$). The function $f_G : \mathbf{B}^n \to \mathbf{B}$ represented by an OKFDD G over X_n is defined as:

1. If G consists of a single node labeled with 0 (1), then G is an OKFDD for $f = 0$ ($f = 1$).
2. If G has a root v with label x_i, then G is an OKFDD for

$$\begin{cases} \overline{x}_i f_{low(v)} + x_i f_{high(v)} & : \ d_i \text{ is Shannon} \\ f_{low(v)} \oplus x_i f_{high(v)} & : \ d_i \text{ is positive Davio} \\ f_{low(v)} \oplus \overline{x}_i f_{high(v)} & : \ d_i \text{ is negative Davio} \end{cases}$$

   where $f_{low(v)}$ ($f_{high(v)}$) are the functions represented by the OKFDD rooted at $low(v)$ ($high(v)$).

If at every node in above definition only Shannon decomposition is applied, the OKFDD will be an Ordered Binary Decision Diagram (OBDD) [Bry86]. If only Davio decompositions are applied, the OKFDD will be an Ordered Functional Decision Diagram (OFDD) [KR93]. As it is evident, the OKFDD is the more general decision diagram than both OBDD and OFDD.

Definition 5 A node in an OKFDD is called a *Shannon node* if it is expanded by Shannon decomposition - Equation (1). It is called a *Davio node* if it is expanded by Davio decompositions - Equations (2) or (3).

As mentioned before Shannon (Davio) nodes are only reduced by type S (D) reductions.

Utilizing reductions, it is possible to define canonical representations of functions. The combination of reduction types I and S is well-known for OBDDs. The notion, f_G, is well defined for OBDDs in the sense that $f_{G_1} = f_{G_2}$ iff G_1 and G_2 are equivalent OBDDs. The same is also true for OFDDs if reduction types I and D are used [BDT93]. Using Lemma 1 canonicity can also be proven for OKFDDs:

Theorem 1 Reduced OKFDDs are canonical representations for Boolean functions if the decomposition types are fixed for every variable.

2.2 OKFDDs and (Reduced, Generalized) Kronecker Expressions

OKFDDs are a multi-level data structure for Boolean functions, that in most cases is used in its reduced form. Nevertheless, in the following we also consider OKFDDs that are not fully reduced, i.e. OKFDDs where the reductions of type S and D (see Figure 1) are not always performed. We now show, how 2-level expressions can be computed from OKFDDs. We distinguish single output and multiple output functions.

2.2.1 Single Output Functions

Let G be an OKFDD, reduced or not reduced. The corresponding 2-level AND/XOR based expression is determined as follows: In each node of the OKFDD a Shannon, a positive Davio or a negative Davio decomposition is carried out. We have a closer look at the paths from the root of the OKFDD to the terminal 1. They are called *1-paths.* To each 1-path P of G we associate a uniquely defined product of literals over X_n, called the *term* t_P *of* P*:* Let e be an edge of P originating in a node v with label x_i. Then x_i is in the product t_P, iff v is either a Shannon node or a positive Davio node and the endpoint of e is $high(v)$. $\bar{x}_i$ is in t_P, iff either v is a Shannon node and the endpoint of v is $low(v)$ or v is a negative Davio node and the endpoint of e is $high(v)$. In all other cases the edge e contributes 1 to t_P.

RKROs, KROs and GKROs are now defined as follows:

Definition 6 Let G be an OKFDD. The 2-level representation obtained by XOR-ing all terms is called the *Generalized Kronecker Expression (GKRO)* corresponding to the OKFDD G. (For simplicity, the class of all 2-level AND/XOR expressions defined in this way is also denoted by GKRO.) A GKRO is called *Reduced Kronecker Expression (RKRO)*, iff the corresponding OKFDD is reduced. A GKRO is called *Kronecker Expression (KRO)*, iff the corresponding OKFDD is complete.

It follows easily, that the GKRO obtained from G is a 2-level expression for f_G, the function represented by G. The reduction types guarantee that a node is only deleted, if the function represented at this node is independant of the corresponding variable, i.e. equivalent OKFDDs represent the same function. Thus, equivalent OKFDDs result in equivalent GKROs representing the same function. It is easy to see, that the above definition of KROs is equivalent to the definition given by Sasao in [Sas93b]. (RKROs and GKROs are not considered there!) Obviously, KRO and RKRO as well define subclasses of GKRO.

We directly conclude from the definitions:

Lemma 2 The number of 1-paths in an OKFDD for a single output Boolean function f is equal to the number of terms in the corresponding GKRO.

Obviously, the number of 1-paths in an OKFDD and thus the number of terms in the corresponding GKRO can be computed in time linear in the size of the OKFDD by a simple DFS algorithm. The effective construction of the GKRO from a given OKFDD has running time $O(n \cdot |terms|)$, where $|terms|$ denotes the number of terms. We illustrate the above definitions with a small example.

Example 1 A reduced OKFDD for a function f is shown in Figure 2, where the left (right) outgoing edge at each node v denotes $f_{low(v)}$ ($f_{high(v)}$). Computation of the 1-paths leads to 4 terms defining the following RKRO for f:

$$f = x_1 x_2 \bar{x}_3 \oplus x_1 x_2 x_4 \oplus x_1 \bar{x}_3 \oplus \bar{x}_1 x_2 x_4$$

A complete OKFDD G' for the same function f can be obtained by inserting two negative Davio nodes with label x_3 and two new Shannon nodes with label x_4. (For illustration see Figure 2.) Computation of the 1-paths then results in a KRO of f with 6 terms, 2 terms more than for the RKRO:

$$\begin{aligned} f &= x_1 x_2 \bar{x}_3 x_4 \oplus x_1 x_2 \bar{x}_3 \bar{x}_4 \\ &\quad \oplus x_1 x_2 x_4 \\ &\quad \oplus x_1 \bar{x}_3 x_4 \oplus x_1 \bar{x}_3 \bar{x}_4 \\ &\quad \oplus \bar{x}_1 x_2 x_4 \end{aligned}$$

The preceding example shows that the RKRO of a function may be smaller than the KRO. (Results for larger examples in Section 4 have shown a significant improvement.)

We finish this subsection with some remarks useful for the computation of minimum or nearly minimal RKROs, KROs and GKROs of a single output function. At first note, that it is sufficient to consider only *reduced* OKFDDs for the computation of good GKROs, since the application of reductions never increases the number of 1-paths: Reductions of type I and D leave the number of 1-paths unchanged. The application of a reduction of type S at a node v reduces the number of paths ending in the successor of v by a factor of $1/2$.

Lemma 3 The minimum GKRO for a single output function is given by the minimum RKRO for this function.

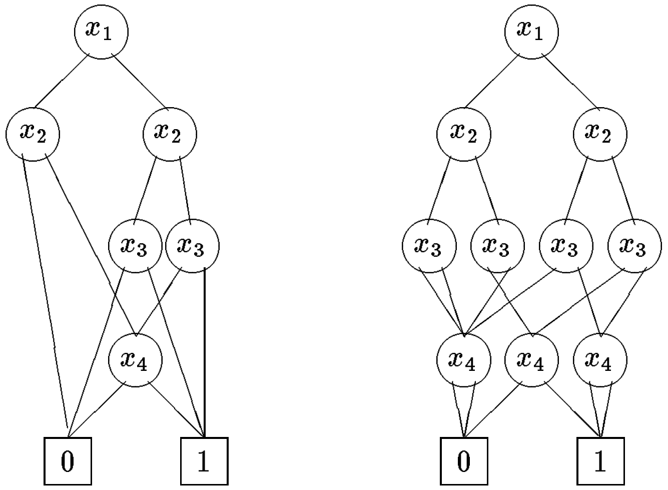

Figure 2: Example for OKFDD and complete OKFDD. x_1 and x_4 are Shannon nodes. x_2 (x_3) are positive (negative) Davio nodes.

For the computation of KROs we do not need to explicitly consider complete OKFDDs. Let G' be the reduction of a given complete OKFDD G. Then it follows easily that the number of 1-paths in G (and even the terms in the corresponding KRO) can be computed by analysis of G'. (For an illustration see again Figure 2.)

Furthermore, it can be shown that reduced (complete) OKFDDs for f with different orderings but unchanged decomposition types define the same RKRO (KRO). This is not true, if the decomposition types for the variables are changed.

If an OKFDD with fixed decomposition types is given and the decomposition type of one variable x_i is to be changed, this can be done efficiently in polynomial time as follows: An XOR-operation is carried out at each node labeled with x_i. In [DST$^+$94] it has been proven that the algorithm for the XOR-operation has worst case running time $O(|G|^2)$, where $|G|$ denotes the size of the OKFDD.

Thus, the computation of an RKRO, KRO or GKRO with a minimum or nearly minimal number of terms for a given single output function f can be performed by the construction of reduced OKFDDs for all or at least many different choices of the decomposition types. Algorithms for the minimization of the number of 1-paths and thus for the minimization of RKROs, KROs and GKROs are presented in the Section 3.

2.2.2 Multiple Output Functions

We now discuss some problems related to the computation of RKROs, KROs and GKROs for multiple output functions using OKFDDs. The three classes are defined analogously for multiple output functions $f = (f_1, f_2, \ldots, f_m)$ as for the case of single output functions: An OKFDD G_j for each component function f_j is used to define a *shared OKFDD* representation G of f. The decomposition type for the variables are fixed over all G_j's. Each G_j defines a GKRO

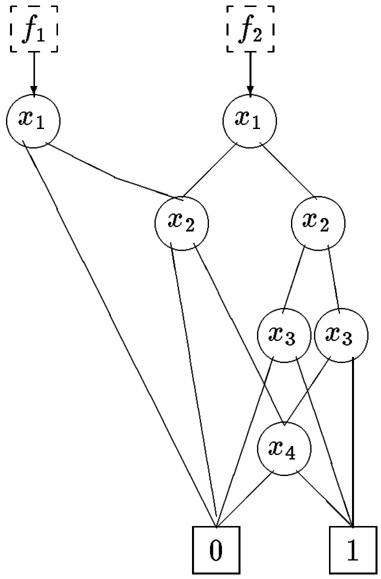

Figure 3: Shared OKFDD with $|terms| \neq |1 - paths|$. x_1 and x_4 are Shannon nodes. x_2 (x_3) are positive (negative) Davio nodes.

for the component function f_j. The GKRO for the whole function f defined by the OKFDD representation G is then obtained by identification of common terms in the GKROs of the component functions, i.e. the union of the terms for all $f_j' s$ defines the set of terms for f. The GKRO has minimum size, if the union has minimum size compared to the sets of terms obtained by any other OKFDD for f. The same definitions apply to RKROs (KROs), if we restrict our attention to reduced (complete) OKFDDs.

In contrast to single output functions Lemma 2 is no longer valid, if multiple output functions are considered. Look at the OKFDD in Figure 3 for an example where the number of terms (= 4) is less then the number of 1-paths (= 5).

Thus, the number of 1-paths is only an upper bound for the number of terms. If we want to determine the exact cardinality of the set of terms, another algorithm than in the single output case has to be applied. A trivial possibility to do this, would be to compare path by path and thus determine the set of terms. But this algorithm is infeasible also for small examples, due to its large runing time.

We point out how this task can be solved much more efficiently with the help of DDs: Let G be a shared OKFDD for $f = (f_1, f_2, \ldots, f_m)$. Then each Shannon node v in G with label x_i is replaced by two nodes, one with label x_i and one with label $\bar{x}_i$, as given in Figure 4. The resulting DD is the representation of m combinational sets $T_1, T_2, \ldots T_m$ over $\{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$, if the DD is interpreted as zero-suppressed OBDD as introduced in [Min93]. The combinational set T_j represents exactly the set of terms for the GKRO of function f_j. For our purposes it is now sufficient to compute the union $T_1 \cup T_2 \cup \ldots \cup T_m$, which is the set of terms for f. Union can be computed efficiently by a simple recursive algorithm on the set representations [Min93].

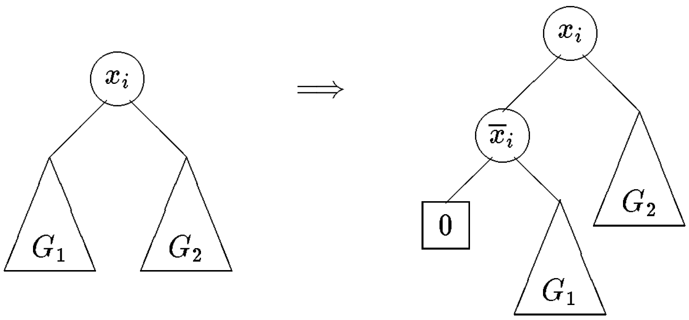

Figure 4: Transformation to zero-suppressed OBDD.

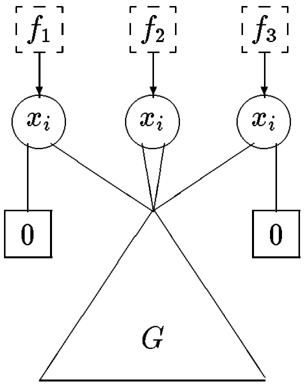

Figure 5: Shared OKFDD whose reduction defines a larger GKRO. x_i's are Shannon nodes.

Furthermore, Lemma 3 is no longer valid for multiple output functions: It may happen that a non-reduced (or even complete) shared OKFDD G defines a smaller GKRO than the reduction of G. Look at Figure 5 for an example. Thus, a minimum RKRO is not necessarily a minimum GKRO, it may even be larger than the minimum KRO, but our experimental results (see Section 4) have shown that this case seems to be of no practical relevance.

3 Minimization of RKROs and KROs

In this section we make use of the close relation between OKFDDs and 2-level AND/XOR expressions as outlined in Subsection 2.2.

3.1 Exact Minimization

We first present an algorithm to determine the minimum RKRO for a given (single or multiple output) function f. For this we build up a reduced OKFDD consisting only of Shannon nodes.

```
exact_minimization (benchmark) {
 okfdd = construct_shannon_okfdd (benchmark) ;
 minGKROsize = number_of_terms (okfdd) ;
 for (all decompositions dec enumerated) {
     okfdd = set_new_polarities (okfdd, dec) ;
     GKROsize = number_of_terms (okfdd) ;
     if (GKROsize < minGKROsize)
        minGKROsize = GKROsize ;
 }
 return minGKROsize ;
}
```

Figure 6: Minimization algorithm

Then this OKFDD is transformed step by step to an OKFDD only with negative Davio nodes. We construct OKFDDs for each possible choice of decomposition types, i.e. we construct 3^n reduced OKFDDs. For each OKFDD we determine the number of terms by the corresponding algorithm from Section 2 (depending on single or multiple output functions). Finally we store the best result. A sketch of the algorithm is given in Figure 6.

A completely analogous algorithm can be used for the computation of the minimum KRO. Only the procedure *number_of_terms* has to be adapted to compute the number of 1-paths in an equivalent complete OKFDD.

Although the algorithm is conceptually very simple, our experimental results show that it has good performance. One main reason for this is the existence of efficient operations on the OKFDDs (and the zero-suppressed OBDDs). The construction of the OKFDDs can be done for (some) functions with several hundred variables. Thus, our approach is mainly limited by the running time, but not by the space requirement.

3.2 Heuristical Minimization

For functions with larger number of variables, i.e. $n > 15$, it is often not feasible (within given time bounds) to compute the best solution. Therefore we use heuristics based on the following strategies

H_{greedy}: Start with a given OKFDD and change the decomposition type of variable x_i ($i \in \{1,..,n\}$) to all possible types. If a new decomposition type decreases the bound for the number of terms, fix this decomposition and go to the next variable (if it exists). Otherwise switch back to the original decomposition and go to the next variable. A sketch of the algorithm is given in Figure 7. Apply this algorithm until no further improvement is obtained.

```
HKRO_minimization (okfdd) {
  size = bound_of_terms (okfdd) ;
  for (i=0; i < no_of_variables; i++) {
      okfdd = switch_decomposition_of_variable (okfdd, i) ;
      if (bound_of_terms (okfdd) < size)
        size = bound_of_terms (okfdd) ;
      else
        okfdd = switch_back_decomposition_of_variable (okfdd, i) ;
  }
  return size ;
}
```

Figure 7: Algorithm of heuristical minimization H_{greedy}

H_{grad}: Start with a given OKFDD and change the decomposition type of variable x_i ($i \in \{1,..,n\}$) to all possible choices. Switch back to the original decomposition and go to the next variable to test the decomposition types of this variable. The best result over all choices finally is choosen. Apply this algorithm until no further improvement is obtained.

Heuristic H_{greedy} tries to get the number of terms down as fast as possible. On the other hand heuristic H_{grad} searches the position, where the largest gain is obtained using a single decomposition change. Notice, that H_{greedy} and H_{grad} as well can be used with a procedure *bound_of_terms*, that not necessarily computes the exact number of terms. Sometimes it is more useful to only compute an upper bound, e.g. by counting the number of 1-paths in linear time.

As pointed out before, for single output functions the minimum RKRO and minimum GKRO are identical. Thus, small RKROs are guaranteed to be small GKROs. Furthermore, the results for RKROs are always better than corresponding results for KROs. The situation is more complicated, if we consider multiple output functions. If minimum GKROs are to be computed, a specific minimization algorithm for GKROs, which considers all the OKFDDs "between" the reduced OKFDD and the corresponding equivalent complete OKFDD, has to be constructed. This is not the topic of this paper. Nevertheless, we point out that the "reduction potential" of OKFDDs can also be used for further optimization of KROs for multiple output functions. We developed an improved KRO-algorithm, that starting from a shared reduced OKFDD computes the union of terms in the following manner. Each time when the union of sets corresponding to OKFDD nodes with differing labels is to be computed, we perform inverse Shannon reductions, until the labels are equal, and after this normal set union with zero-suppressed OBDDs is performed. The resulting 2-level expression is guaranteed to be smaller ($\leq$) than the corresponding KRO. Normally it is a GKRO, which is neither an RKRO nor a KRO. In all our experiments the resulting GKRO was indeed smaller than the corresponding KRO, but in general we did not obtain the quality of the RKROs. We therefore consider minimization of RKROs to be a good compromise between computational complexity and quality of the solution also for the class GKRO.

name	*in*	*out nr*	*FPRM*	*KRO*	*RKRO*
mlp4	8	4	24	24	21
log8mod	8	5	26	22	16
dist	8	4	87	60	40
dist	8	5	93	82	52
sym10	10	1	266	266	193
sao2	10	2	55	40	23
Z9sym	9	1	173	173	130

Figure 8: Minimal KROs and RKROs for single output functions

name	*in*	*out*	*FPRM*	*KRO*	*RKRO*	*PSDKRO*
mlp4	8	8	97	97	85	81
dist	8	5	185	157	126	105
sao2	10	5	100	76	63	34
add6	12	7	132	132	132	132

Figure 9: Minimal KROs and RKROs for multiple output functions

4 Experimental Results

In this section we present experimental results that were carried out on the OKFDD package presented in [DST$^+$94] on a *Sun Sparc 1+* workstation.

4.1 Exact Minimization

We applied our minimization algorithm to several single and multiple output functions of MCNC benchmarks. The results can be found in Figure 8 and 9. *name* denotes the name of the circuit. *in* denotes the number of inputs and *out nr* (*out*) denotes the number of the considered output (outputs). In column *FPRM* the number of terms of the minimal Fixed Polarity Reed-Muller Expression [DTB94] is given to show the improvement obtained by the consideration of more general classes. The minimal number of terms for the KRO and RKRO are given in columns *KRO* and *RKRO*, respectively.

These first experiments show that often RKROs and KROs allow much more efficient representation of Boolean functions than the class FPRM. In the last column we compare our results with the class of Pseudo Kronecker Expressions (PSDKRO) [Sas93b]. The results show that PSDKRO allow in general smaller representation, but no efficient minimization algorithm for larger examples is known. For all considered benchmarks the minimal RKRO is more efficient than the minimal KRO.

name	KRO			RKRO			GKRO	
	min	H_{greedy}	H_{grad}	min	H_{greedy}	H_{grad}	H_{greedy}	H_{grad}
mlp4	97	97	97	85	85	85	92	92
Z9sym	173	173	173	130	133	133	133	133
dist	157	164	187	126	126	126	146	146
sym10	266	306	306	193	193	193	193	193
sao2	76	76	81	63	66	81	63	57
add6	132	137	132	132	132	132	132	132

Figure 10: Comparison of minimal and heuristical results

name	*in*	*out*	KRO		RKRO		GKRO	
			H_{greedy}	H_{grad}	H_{greedy}	H_{grad}	H_{greedy}	H_{grad}
chkn	29	7	690	690	243	243	285	285
mish	94	43	-	-	53	53	-	-
cps	24	109	251	253	214	217	173	173
apex7	48	37	-	-	302	234	-	-
soar	83	94	-	-	476	477	-	-
vg2	25	8	4098	4474	484	488	1132	1176
s1196	32	32	5441	4956	2333	2360	4049	3845
cordic	23	2	3264	1980	1943	1301	1948	1302
bc0	26	11	467	467	349	338	314	314
apex5	117	88	-	-	509	509	-	-
x2dn	82	56	-	-	125	125	-	-

Figure 11: Heuristical minimization of large functions

4.2 Heuristical Minimization

We first apply the heuristics H_{greedy} and H_{grad} to all benchmarks for which the minimal number of terms is known. The results for each type of Kronecker expression is given in Figure 10. Columns for GKRO indicate results obtained for the improved KRO-algorithm. For benchmark *sao2* heuristic H_{grad} obtained a GKRO that is smaller than the optimal KRO and the optimal RKRO. As can be seen from the table the results obtained by the heuristics are very good compared with the minimal number of terms.

The heuristics were also applied to larger circuits. The results are given in Figure 11. (A - at a position symbolizes, that the program was not able to compute the initial number of paths within 10 CPU seconds.)

In general the results obtained by both heuristic types are comparable. In two examples (apex7, corcic) H_{grad} is significantly better. On the other hand, the running time of H_{grad} is much longer, since in each iteration at most one decomposition type is changed.

The best overall results were obtained for RKROs. Additonally, the minimization of this class is performed the fastest, since all computations are carried out on reduced OKFDDs. This speeds up the computation tremendously, especially for functions with a very large number of inputs.

5 Conclusions

We presented two new classes of 2-level AND/XOR expressions, called Reduced Kronecker Expression and Generalized Kronecker Expression, and methods for their exact and heuristical minimization. For this we used Ordered Kronecker Functional Decision Diagrams, that are a closely related data structure.

We presented exact and heuristical methods for minimization of large functions with more than 100 variables. We compared the new classes to the well-known classes of Fixed-Polarity Reed-Muller Expressions and (Pseudo) Kronecker Expressions. Experimental results were given to show the efficiency of our approach.

Our methods can be generalized to Pseudo Kronecker Expressions, if Free Kronecker Functional Decision Diagrams with compatible types [BD94a] are considered instead of OKFDDs. It is an interesting open problem how to find "good types" resulting in small 2-level expressions.

References

[BD94a] B. Becker and R. Drechsler. Efficient graph based representation of multi-valued functions with application to genetic algorithms. In *Proceedings of the 24nd Int. Symp. of Multi-Valued Logic*, 1994.

[BD94b] B. Becker and R. Drechsler. Testability of circuits derived from functional decision diagrams. In *European Conf. on Design Automation*, page 667, 1994.

[BDT93] B. Becker, R. Drechsler, and M. Theobald. On the implementation of a package for efficient representation and manipulation of functional decision diagrams. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[Ber68] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill Book Company, 1968.

[BFRV92] S.D. Brown, R.J. Francis, J. Rose, and Z.G. Vranesic. *Field-Programmable Gate Arrays.* Kluwer Academic Publisher, 1992.

[BHMS84] R.K. Brayton, G.D. Hachtel, C. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Cluwer Academic Publishers, 1984.

[Bry86] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 8:677–691, 1986.

[CFM93] O. Coudert, H. Fraisse, and J.-C. Madre. A breakthrough in two-level logic minimization. In *Int'l Workshop on Logic Synth.*, page P2b, 1993.

[DB93] R. Drechsler and B. Becker. Rapid prototyping of fully testable multi-level and/exor networks. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[DST+94] 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.*, 1994.

[DTB94] R. Drechsler, M. Theobald, and B. Becker. Fast FDD based minimization of generalized reed-muller forms. *22. Workshop über Komplexitätstheorie, Datenstrukturen und effiziente Algorithmen, Saarbrücken*, 1994.

[FTY87] H. Fleisher, M. Tavel, and J. Yeager. A computer algorithm for minimizing reed-muller canonical forms. In *IEEE Transaction on Computers*, volume c-36, pages 247–250, 1987.

[GM94] J. Gergov and C. Meinel. Efficient analysis and manipulation of obdds can be extended to read-once-only branching programs. to be published in *IEEE Trans. on Comp.*, 1994.

[KR93] U. Kebschull and W. Rosenstiel. Efficient graph-based computation and manipulation of functional decision diagrams. In *Proceedings of EDAC'93*, pages 278–282, 1993.

[Min93] S. Minato. Zero-suppressed BDDs for set manipulation in combinational problems. In *Design Automation Conf.*, pages 272–277, 1993.

[Pap79] G. Papakonstantinou. Minimization of modulo-2 sum of products. In *IEEE Transaction on Computers*, volume c-28, pages 163–167, 1979.

[PC90] M.A. Perkowski and M. Chrzanowska-Jeske. An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified boolean functions. In *Int'l Symp. Circ. and Systems*, pages 1652–1655, 1990.

[PCSS92] M. A. Perkowski, L. Csansky, A. Sarabi, and I. Schäfer. Fast minimization of mixed-polarity and/xor canonical networks. In *Proceedings of ICCD'92*, pages 33–36, 1992.

[Per92] M.A. Perkowski. The generalized orthonormal expansion of functions with multiple-valued inputs and some of its application. In *Proceedings of the 22nd Int. Symp. of Multi-Valued Logic*, pages 442–450, 1992.

[Red72] S. M. Reddy. Easily testable realizations for logic functions. In *IEEE Transaction on Computers*, volume c-21, pages 1183–1188, 1972.

[Rol93] U. Rollwage. The complexity of mod-2 sum PLA's for symmetric functions. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[Sas93a] 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.

[Sas93b] T. Sasao. *Logic Synthesis and Optimization.* Kluwer Academic Publisher, 1993.

[Sau92] J. Saul. Logic synthesis for arithmetic circuits using the reed-muller representation. In *Proceedings of EDAC'92*, pages 109–113, 1992.

[SB90] T. Sasao and Ph. Besslich. On the complexity of mod-2 sum PLAs. *IEEE Trans. on Comp.*, 39:262–266, 1990.

[SP91] I. Schäfer and M.A. Perkowski. Multiple-valued input generalized reed-muller forms. In *Proceedings of the 21nd Int. Symp. of Multi-Valued Logic*, pages 40–48, 1991.

[SP92] A. Sarabi and M. A. Perkowski. Fast exact and quasi-minimal minimization of highly testable fixed-polarity and/xor canonical networks. In *Proceedings of DAC'92*, pages 30–35, 1992.

[SP93] A. Sarabi and M.A. Perkowski. Design for testability properties of and/xor networks. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[SR75] K. K. Saluja and S. M. Reddy. Fault detection test sets for reed-muller canonical networks. In *IEEE Transaction on Computers*, volume c-24, pages 995–998, 1975.

[SW93] D. Sieling and I. Wegener. Reduction of BDDs in linear time. *Information Processing Letters*, 48(3):139–144, 11 1993.

[TM93] C.C. Tsai and M. Marek-Sadowska. Efficient minimization algorithms for fixed polarity and/xor canonical networks. In *Great Lakes Symp. VLSI*, pages 76–79, 1993.