Fast Heuristic Minimization of Exclusive-Sums-of-Products*

Alan Mishchenko and Marek Perkowski
Department of Electrical and Computer Engineering
Portland State University, Portland, OR 97207, USA
[alanmi,mperkows]@ee.pdx.edu

Abstract

Exclusive-Sums-Of-Products (ESOPs) play an important role in logic synthesis and design-for-test. This paper presents an improved version of the heuristic ESOP minimization procedure proposed in [1,2]. *The improvements concern three aspects of the procedure: (1) computation of the starting ESOP cover; (2) increase of the search space for solutions by applying a larger set of cube transformations; (3) development of specialized data-structures for robust manipulation of ESOP covers. Comparison of the new heuristic ESOP minimizer EXORCISM-4 with other minimizers (EXMIN2* [3], *MINT* [4], *EXORCISM-2* [1] *and EXORCISM–3* [2]*) show that, in most cases, EXORCISM-4 produces results of comparable or better quality on average ten times faster.*

Introduction

Two-level Sums-Of-Products (SOPs) have been widely used to represent and manipulate Boolean functions. Exact and heuristic SOP minimization has traditionally attracted attention of researchers because in many applications it is important to have as compact SOP representation as possible. The value of Exclusive-Sums-Of-Products (ESOPs) and ESOP minimization, on the other hand, has been underestimated for the following reasons:

- The EXOR gate, as implemented in the widely used standard cell libraries, is almost two times larger and slower compared to NAND and NOR gates.
- Relatively few algorithms use ESOPs for internal representation of Boolean functions.
- The exact minimization of ESOPs is practical only for arbitrary functions of five input variables and special classes of functions up to ten variables [5], while the heuristic minimization is based on search algorithms and is computationally more expansive than SOP minimization.

However, the EXOR gate and ESOPs play an important role in logic synthesis, design for test, and other areas of computer technology:

- For many Boolean functions, the number of cubes in minimal ESOPs is less then the number of cubes in minimal SOPs [6].
- The EXOR gate has excellent testability properties leading to efficient methods for automatic test pattern generation and design for test [7].
- For many types of practical circuits, the selective use of EXOR gates in logic synthesis yields better implementations in terms of both area and delay [8,9].

Over the years, several ESOP minimization strategies and ESOP minimizers have been created but none of them matched well with our research purposes. Some of the tools did not belong to the public domain; others had limitations on the input data format. Above all, the available tools had performance limitations when applied to large Boolean functions. This motivated our work on a new user-friendly minimizer EXORCISM-4, which follows the tradition of previous ESOP minimizers, EXORCISM-2 [1] and EXORCISM-3 [2], developed at Portland State University.

This paper presents the results of research directed towards the development of a new ESOP minimization tool. We have experimented with a number of approaches and algorithms and selected those that proved to be practically efficient. In our presentation below, both the successful and the unsuccessful attempts to improve the performance of ESOP minimization will be mentioned.

The rest of the paper is organized as follows. Section 1 discusses the previous work in ESOP minimization. Section 2 gives the main definitions. Section 3 discusses the general data-flow in our minimization procedure. Section 4 reviews the starting cover computation. Section 5 introduces and illustrates the ExorLink operation. Section 6 presents the

* This work was supported by a research grant from Intel Corporation.

look-ahead strategies. Section 7 discusses data structures and implementation issues. Section 8 gives experimental results comparing the new tool with other tools. Section 9 concludes the paper and outlines the future research directions.

1 Previous Work

Exclusive-Sums-of-Products have been introduced by I. I. Zhegalkin in 1927 [10,11] and later independently rediscovered by S. M. Reed [12] and D. E. Muller [13]. A systematic classification of various families of Reed-Muller (Zhegalkin) expresions has been given in [14].

The approaches to heuristic ESOP minimization can be roughly classified as follows:

- Iterative cube transformation [1-4,15-18]
- Minimization of families of ESOP expansions [19-21]
- Simplification based on term rewriting [22]
- Zakrevskij stair-case method [23]

Most of the ESOP minimizers use the first strategy. They iteratively apply a set of selected cube transformations to single cubes, cube pairs, or larger cube groups. If replacement of the starting cubes by the resulting cubes leads to simplification or some other desirable improvement, the cover is modified without changing the function represented by it.

The approach developed in this paper also uses cube transformations. The difference is that we use one but very general cube transformation, *distance-k ExorLink*. When applied in the context of limited look-ahead discussed in Section 7, ExorLink subsumes many other (possibly all) cube transformations known to date.

Relevant to our approach (in particular, to the computation of a starting cover) is the second minimization approach listed above. This approach attempts heuristic ESOP minimization by developing efficient algorithms to minimize specialized classes of AND-EXOR expressions: Positive Polarity Reed-Muller Forms (PPRMs), Kronecker and Pseudo-Kronecker Forms (KRMs and PKRMs), Generalized Reed-Muller Forms (GRMs), etc. These classes differ from ESOPs in restrictions applied to polarities of literals or in the type of canonical expansions used to the generic decision diagrams. The reader is referred to [6,14] for definitions of these classes.

Because the set of all ESOPs is a superset of expressions belonging to any of the above mentioned classes, an exact or heuristic minimization procedure developed for a class of functions may be used as an approximation, or a starting point, for ESOP minimization.

Of particular importance to the research in this paper is the algorithm for exact minimization of Pseudo-Kronecker expressions for symmetric functions [21]. This algorithm gives a good trade-off between the quality and the runtime. On the one hand, Pseudo-Kronecker forms are a relatively comprehensive subclass of ESOPs and therefore finding the exact minimum in this class yields relatively good quality. On the other hand, the Pseudo-Kronecker minimization algorithm exploits the BDD representation and therefore can be applied to a wide range of benchmarks, for which shared BDDs can be constructed. This algorithm is discussed in detail in Section 4.

2 Preliminaries

This section gives the basic definitions used in the paper.

The truth tables and Karnaugh maps of OR and Exclusive-OR (EXOR) operations are shown in Fig. 1.

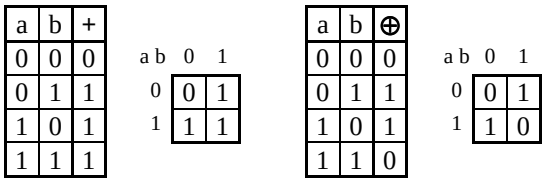

Figure 1. OR and EXOR operations.

A *literal* is a Boolean variable in the negative or positive polarity. A *cube* is a product term composed of literals using Boolean AND operation.

Two cubes *coincide in variable* x if x does not appear in the cubes or if x appears in the cubes in the same polarity. Two cubes *differ in variable x* if they do not coincide in variable x. The *distance* between two cubes is the number of variables, in which the cubes differ.

A variable appears in the cube in one of the three *forms*: (1) negative polarity; (2) positive polarity; (3) don't-care.

For example, assuming that cube $a\bar{b}\,\bar{d}$ belongs to function with input variables (a,b,c,d), variable a appears in positive polarity, variables b and d appear in negative polarity, and variable c appears as a don't-care.

An *Exclusive-Sum-Of-Produces (ESOP)* is an Exclusive-OR of zero or more cubes. An ESOP is *reduced* if it does not contain identical cubes. An ESOP is *minimal* if all cube pairs have distance 2 or more. An ESOP is *exact minimum* if it contains the minimum number of cubes among all ESOPs representing the given function.

The following propositions can be proved using the fundamental property of EXOR operation.

Proposition 1: Two identical cubes (distance-0 cubes) can be added to any ESOP without changing the function represented by it.

Proposition 2: The EXOR of two cubes that have distance 1 can be represented by a single cube.

The ExorLink operation discussed in Section 5 extends these two propositions to cube pairs with arbitrary distance between the cubes.

3 ESOP Minimization Algorithm

This section outlines the main steps of the heuristic ESOP minimization procedure and data representations used to implement these steps.

Minimization Algorithm

The pseudo-code of the minimization algorithm is shown in Fig. 2. The minimization procedure takes the multi-output function and the quality parameter, which determines how many minimization loops are performed.

```
esop HeuristicMinimization( func F, quality Q )
{
   esop Cover = GenerateStartingCover( F );
   while ( Q > 0 ) {
     ResetCubePairs( Cover );
     do {
         do {
            Cover= AgressiveMinimization(Cover);
         } while (there is improvement);
         Cover= LastGaspMinimization(Cover);
      } while (there is improvement);
     Q = Q-1;
   }
    Cover = RefinementMinimization(Cover);
    return Cover;
}
```

Figure 2. Pseudo-code of ESOP minimization algorithm.

After computing the starting cover using the BDD-based algorithm [21], the data flow enters the main minimization loop. Each time the main loop is entered internal data structures storing the candidate pairs for the application of distance-2, distance-3 and distance-4 Exorlink are reset.

Inside the main loop, there is another loop that applies aggressive minimization strategy, as described in section 6, “Look-Ahead Strategies”. When aggressive minimization does not lead to improvement, the last gasp is applied, which tries to get the cover out of the local minimum by several additional rounds of distance-4 ExorLink.

Finally, when the minimization loops terminate, there is a call to the procedure RefinementMinimization() which tries to improve the literal count of the cover without attempting to reduce the number of cubes.

This structure of the minimization procedure was found as a result of extensive experimentation having the goal of finding a sequence of minimization operations that perform relatively well on a wide selection of benchmarks.

Data Representation

The input to the ESOP minimization software is the text file in standard PLA or BLIF formats, representing a multi-output Boolean function. In the current version of the minimizer, the don’t-cares of the input function are ignored and only the on-set of the function is considered.

The processing starts by creating the shared BDDs for the input function. Dynamic reordering of variables is enabled during the BDD construction, which allows us to work with relatively large multi-output functions.

In general, the minimizer uses mixed explicit and implicit data representations. After computing the starting cover using shared BDDs, the representation switches to cubes encoded in positional notation and stored explicitly using bit strings. The subsequent cube operations and cover transformations are carried out using the explicit representation, except the verification step at the end. During verification, the resulting cover is again converted into shared BDDs and compared with the BDDs constructed for the input function.

We experimented trying to replace the explicit representation of cubes as bit strings by their implicit representation using Zero-Suppressed Binary Decision Diagrams (ZDDs) [26]. This attempt did not succeed because ZDDs speed-up computation only if there is a way to process many cubes in parallel. Meanwhile cube transformations used in our approach, in particular, finding all distance-k cube pairs and performing ExorLink operation, requires processing cubes one at a time.

4 Starting Cover Computation

This section summarizes the BDD-based algorithm for efficient minimization of the Pseudo-Kronecker expressions used to compute the starting cover. A detailed discussion of the algorithm, including the proof that it computes the exact minimum of the Pseudo-Kronecker expressions for symmetric functions, is given in [21].

The algorithm works in two passes. During the first pass, the shared BDD of the input function is traversed depth-first and in each node the best expansion is found and saved in the lossless cache. The expansions are selected out of the three canonical expansions (Shannon, Positive Davio or Negative Davio) and the number of cubes they add to the solution is used to evaluate their cost. It is possible to count the number of cubes in the resulting Pseudo-Kronecker expressions because they result from “flattening” the Pseudo-Kronecker decision diagrams, which are simulated (not explicitly constructed) by the algorithm presented in Fig. 3. This algorithm also minimizes the number of literals in the minimal Pseudo-Kronecker expression by preferring Positive or Negative Davio to Shannon expansion whenever it does not increase the cube count.

During the second pass, the enumeration of all paths in the diagram is performed (Fig. 4). In each node, the best expansion is retrieved from the lossless cache and, depending on the expansion, one literal is added to the

array of the variable values representing the current cube. When the traversing procedure reaches the bottom of the diagram, it has the values of all variables collected in the array, which is now used to generate the cube and add it to the resulting cover. At the end of the traversal, the resulting cover contains all cubes belonging to the exact minimum of the Pseudo-Kronecker expression.

```
(exptype,int) CountCubesInExactPseudoKro( func F )
{
   extype Exp; int Cost;

   // consider the terminal cases
   if ( F == 0 )   return ( pDavio, 0 );
   if ( F == 1 )   return ( pDavio, 1 );
   if ( ( Exp, Cost ) = CheckCacheForResult( F ) )
      return ( Exp, Cost );

   // determine the cofactors
   ( F0, F1 ) = DecomposeBdd( F, TopVar(F) );

   // recursively solve subproblems
   int N0, N1, N2, Nmax, Cost;
   N0 = CountCubesInExactPseudoKro( F0 );
   N1 = CountCubesInExactPseudoKro( F1 );
   N2 = CountCubesInExactPseudoKro( F0 ⊕ F1 );

   // determine the most costly expansion
   MaxN = max( N0, N1, N2 );

   // choose the least constly expansion
   if ( MaxN == N0 )
       Exp = pDavio;  Cost = N1 + N2;
   else if ( MaxN == N1 )
       Exp = nDavio;  Cost = N0 + N2;
   else /* if ( MaxN == N2 ) */
       Exp = Shannon; Cost = N0 + N1;

   // cache and return the result
   InsertIntoCache( F, Exp, Cost );
   return ( Exp, Cost );
}
```

Figure 3. A procedure for counting the number of cubes in minimal Pseudo-Kronecker expansions.

5 ExorLink Operation

ExorLink operation has been introduced in [24] as a generalization of several simple cube transformations used in various approaches to ESOP minimization. The main idea of ExorLink is to replace two cubes by a set of cubes without changing the function.

The number of cubes to be substituted instead of the initial cube pair is equal to the distance between the starting cubes. If the distance between the cubes is more than 1, there are several possibilities of performing ExorLink, each of them leading to functionally equivalent representation composed of different cubes.

For example, consider the following equivalent transformations of cubes $a\bar{b}$ and $\bar{a}b$ constituting the distance-2 ExorLink operation:

```
GenerateExactPseudoKro( bdd F, char * VarValues )
{
   // consider the terminal cases
   if ( F == 0 )   return;
   if ( F == 1 ) {
       cube NewCube = CreateCube( VarValues );
       AddCubeToCover( NewCube );
       return;
   }

   // find the best expansion by a cache lookup
   extype Exp = CheckCacheForResult( F );

   // determine the top-most variable
   var X = TopVar(F);

   // determine the cofactors
   ( F0, F1 ) = DecomposeBdd( F, X );

   // generate cubes in the left/right branches
   if ( Exp = pDavio ) {
       VarValues[ X ] = VAR_ABSENT;
       GenerateExactPseudoKro( F0, VarValues );
       VarValues[ X ] = VAR_POSITIVE;
       GenerateExactPseudoKro( F0⊕F1, VarValues);
   }
   else if ( Exp = nDavio ) {
       VarValues[ X ] = VAR_ABSENT;
       GenerateExactPseudoKro( F0, VarValues );
       VarValues[ X ] = VAR_NEGATIVE;
       GenerateExactPseudoKro( F0⊕F1, VarValues);
   }
   else /* if ( Exp = Shannon ) */ {
       VarValues[ X ] = VAR_NEGATIVE;
       GenerateExactPseudoKro( F0, VarValues );
       VarValues[ X ] = VAR_POSITIVE;
       GenerateExactPseudoKro( F1, VarValues );
   }
}
```

Figure 4. A procedure for the generation of minimal Pseudo-Kronecker expansions.

$$a\bar{b} \oplus \bar{a}b = a \oplus b = \bar{a} \oplus \bar{b}.$$

In general, distance-k ExorLink operation produces k! cube groups containing k cubes each. In all the cube groups there are $k*2^{k-1}$ different cubes, some of them appearing is more than one group. The cube set generation is illustrated by distance-3 ExorLink applied to cubes abc and $\bar{a}\bar{b}\bar{c}$ in Fig. 5. Zeros are not shown.

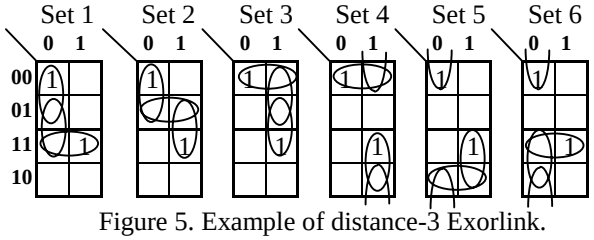

Figure 5. Example of distance-3 Exorlink.

Observe that cube groups correspond to different paths in the Boolean space connecting minterms abc and $\bar{a}\bar{b}\bar{c}$. If the distance between the starting cubes is k, there are k directions to make the first step, k-1 directions to make the second step, and so on. As a result, there are k! paths each of them corresponding to one cube group.

6 Look-Ahead Strategies

The previous ESOP minimization algorithms [1,2] used a combination of full backtracking and limited backtracking. *Full backtracking* means exploring all the branches of the search tree up to a certain depth, selecting the best way to modify the cubes, and performing the required transformation. *Limited backtracking* stands for exploring a fixed number of branches in the search tree up to a certain depth. The branches to explore may be selected randomly or found heuristically.

For ESOP minimization with ExorLink as a basic cube transformation routine, it was found that "large-depth look-ahead with a small number of backtracks is much faster and usually gives better results than the small-depth look-ahead and a large number of backtracks, although the latter is also sometimes very helpful". [25]

We tested these and other look-ahead strategies and finally adopted the one described as "backtrack until the first success". "Success" is defined differently depending on the minimization strategy and the distance between the cubes, to which ExorLink is applied.

Table 1 lists the criteria, which our algorithm uses to determine whether the transformation is successful and should be applied, or whether backtracking should continue by testing the next available cube group. The next available group may result from the same ExorLink (recall that distance-k ExorLink produces k! cube groups) or from ExorLink applied to the next cube pair in the queue of pairs to be considered.

Table 1. The criteria for accepting cube transformations resulting from the ExorLink operation.

ExorLink Distance	Criteria for Accepting Cube Transformations	
	Aggressive Strategy	Refinement Strategy
2	Leads to reduction in the number of cubes.	Does not increase the number of cubes. Leads to reduction in the number of literals.
3	Does not increase the number of cubes.	Does not increase the number of cubes. Leads to reduction in the number of literals.
4	Does not increase the number of cubes.	Not used

In different modes of operation our algorithm applies a different combination of aggressive and refinement minimization. The general rule is that in each minimization loop, aggressive strategy is applied several times to all cube pairs starting from distance-2 ExorLink up to distance-4 ExorLink.

Depending on the requested minimization quality the algorithm performs different number of minimization loops. This number roughly corresponds to the value of the quality parameter specified on the command line by the switch "-q". For example, "-q5" results in five minimization loops.

The trade-off between the quality and the computation time is illustrated in Table 2 using benchmark "alu4.pla", which has 14 inputs and 8 outputs. The shared BDD node count is 804. The input file reading and the starting cover computation time are 0.07 sec and 0.03 sec, respectively. The experiment is performed on a 933MHz Pentium III PC.

Table 2. Quality/runtime trade-off for alu4.pla.

	Quality level ("-qN" switch), N										
	0	1	2	3	4	5	6	7	8	9	10
Cubes	440	440	435	435	415	415	415	414	411	411	411
Time, c	2.1	2.3	5.9	6.8	16.2	16.7	17.1	20.9	21.8	22.1	22.9

Using the above minimization strategy in the context of ExorLink-4 allows us to make the following statement about the minimization quality achieved by the algorithm.

Theorem. If the program runs long enough to completely explore the subspace of each local minimum, the above minimization strategy always escapes from any local minimum that is one or two cubes deep.

Compared to this, distance-3 ExorLink [25] can only guarantee getting out of the local minima one cube deep. This difference explains why in some cases our minimizer has found ESOP covers of smaller cardinality compared to other minimizers (see section "Experimental Results").

The following example illustrates the application of the look-ahead strategies in the heuristic ESOP minimization. Consider the function $F = \bar{a}\,\bar{c} \oplus \bar{a}b\bar{c}\bar{d} \oplus ab \oplus a\bar{c}d$, shown in Fig. 6 (left). The goal is to find the ESOP of this function composed of the three cubes, $\bar{a}\bar{b}\bar{c}\bar{d} \oplus \bar{c}d \oplus ab$, shown in Fig. 6 (right). It is easy to see that the desired cube transformation consists of reshaping two cubes ($\bar{a}\,\bar{c}$, $\bar{a}b\bar{c}\bar{d}$), as shown in Fig. 6 (center), followed by combining two distance-1 cubes ($\bar{a}\,\bar{c}d$, $a\,\bar{c}d$) into one cube, $\bar{c}d$.

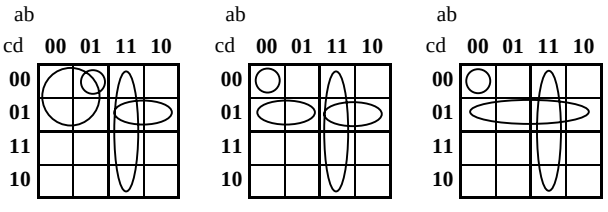

Figure 6. Example illustrating the look-ahead strategies in the heuristic ESOP minimization.

When the look-ahead is applied automatically by the program, it will detect the distance-2 and distance-3 cube pairs and apply the ExorLink operation to them. The resulting cubes will be checked for being distance-0 or distance-1 away from the cubes currently in the cover.

In the example of Fig. 6 (left), suppose the program first applies the ExorLink operation to distance-2 cubes. There is only one pair of distance-2 cubes, $\bar{a}\,\bar{c}$ and $\bar{a}b\bar{c}\bar{d}$, shown in Fig. 7 (left). Two cube groups derived by ExorLink-2 are given in Fig. 7 (center) and Fig. 7 (right). Obviously, the second group will be accepted as the one leading to the reduction in the number of cubes.

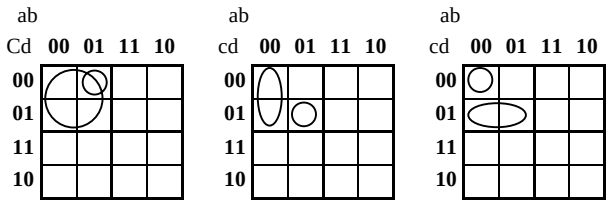

Figure 7. Two cube groups resulting from ExorLink of the distance-2 cubes, $\bar{a}\,\bar{c}$ and $\bar{a}b\bar{c}\bar{d}$.

Suppose the program first applies the ExorLink operation to distance-3 cubes. There are several pairs of distance-3 cubes. Here we discuss only one pair, ab and $a\bar{c}d$. Consideration for other cube pairs is similar. The resulting six cube groups, each composed of three cubes, are given in Fig. 8. Only one cube group features a cube ($ab\bar{c}\bar{d}$) that is distance-1 removed from a cube in the cover ($a\bar{b}\bar{c}\bar{d}$). If this cube group is accepted, then the cover is reshaped but the number of cubes is not reduced. The reduction will be achieved in the future iterations by applying the ExorLink operation to other cube pairs.

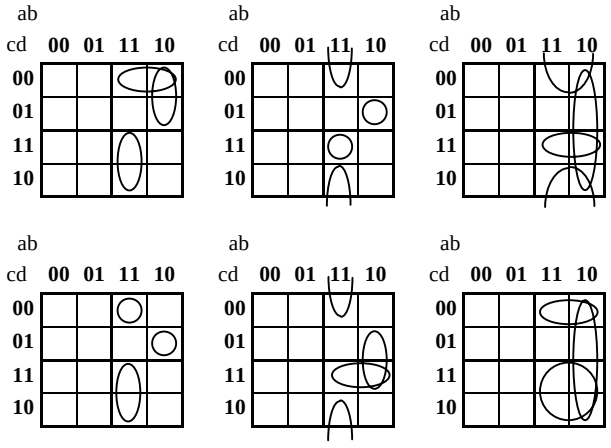

Figure 8. Six cube groups resulting from ExorLink of the distance-3 cubes, ab and $a\bar{c}d$.

7 Implementation Issues

The following general principles of high-performance programming have been used in the implementation of EXORCISM-4.

Allocate Memory With Caution

The heuristic minimization algorithm described in this paper generates a large set of intermediate cubes, which only reshape the cover without improving its cardinality. Because calls to system memory allocation are relatively expensive in terms of runtime, EXORCISM-4 performs efficient memory recycling. Memory allocated at the beginning for the starting cover is never reallocated because the size of the cover cannot increase during minimization. To ensure efficient memory recycling, cubes that are not currently in use are kept in a linked list. Adding and removing cubes in the linked list is much more efficient than calling systems memory allocation whenever a new cube is created or deleted.

Terminate Computation As Early As Possible

In many cases, it is possible to speed up computation by early detection of a situation when the result of computation is useless for whatever reason. The performance improvement is more or less substantial depending on how often this situation occurs and how large is the part of computation that can be skipped.

EXORCISM-4 uses early termination of computation in the several cases.

- One of the operations repeated very often is computing the distance between two cubes. The algorithm considers only cube pairs with distance four or less. It means that as soon as at least five different variables are detected in the cubes, the rest of computation can be skipped. This simple idea gives up to 20% improvement in runtime for benchmarks with many inputs and outputs.
- As shown in Section 6, distance-k ExorLink operation produces $k*2^{k-1}$ new cubes grouped into k! groups in such a way that some of the cubes are encountered in more than one group. Our algorithm considers cube groups one by one and as soon as it find the matching group, this group is used to reshape the cover. Consequently, ExorLink is implemented in such a way that instead of generating all $k*2^{k-1}$ cubes at once, it generates them “on demand”. This trick is responsible for speeding up distance-4 ExorLink by at least 50%, because in practice only 10 cubes on average out of 32 are often enough.

Reuse Results of Previous Computation

The importance of this principle can hardly be overestimated, as witnessed by the success of caching techniques speeding-up computation by many orders of magnitude in BDD-based applications.

In case of EXORCISM-4, the standard BDD caching techniques are used during the starting cover computation. There are two other cases when caching helps reduce unnecessary computation:

- Each new cube generated by ExorLink is tested for distances with all other cubes present in the cover. Even though at the beginning of testing it is not known whether the cube will be taken into the cover, the testing procedure stores information about those cubes currently in the cover that have distances 2, 3, and 4 with the given cube. As a result, if the cube is eventually accepted into the cover, there is no need to compute candidate pairs for distance 2, 3, or 4 ExorLink in the following iterations.
- Finally, there is a way to reduce computation by saving the results of distance-testing for those cubes that are generated by ExorLink and included into more than one out of k! groups.

Use Sorted Data Structures

Using sorted data structures (ordered lists, heaps, tries, Patricia trees, etc.) often leads to faster search and reduction of the access time.

We tried to store the cubes in an ordered list and in an ordered matrix (sorting them by the number of positive and negative literals). The rationale for doing this would be to facilitate finding small-distance cube pairs, and thereby to speed up cube selection for ExorLink. However, in practice it turned out that sorting cubes does not help reducing the runtime, and eventually we switched back to representing the cover as an unsorted linked list cubes.

8 Experimental results

EXORCIMS-4 has been implemented in platform-independent C++ using the BDD package CUDD [27] and tested extensively on Unix and Windows workstations. The experimental results below have been received on a 933MHz Pentium III PC under Microsoft Windows 2000. At all times the program used no more than 50Mb of RAM.

A BDD-based verifier built into EXORCISM-4 has been used to check the correctness of the minimization results for all the benchmark.

To make a fair runtime comparison with earlier programs, the time measurements of our program have been normalized to reflect the speed of computers used to run the programs. In particular, for comparison with EXMIN2 and MINT, the computer used to get experimental result in [4] has been considered 58 time slower then 933MHz Pentium III. In case of EXORCISM-2 and -3, the coefficient was 24. The normalization coefficient was determined by considering benchmark “9sym.pla”.

Table 3 compares the results of minimization by EXORCISM-4 with those of EXMIN2 [3] and MINT [4]. Table 4 compares EXORCISM-4 with EXORCISM-2 [1] and EXORCISM-3 [2]. The comparison is in terms of the number of cubes and literals in the final solution and the CPU time spent for minimization. The last two columns in Tables 3 and 4 give the ratio of the runtime needed by the specified minimizer and the runtime of EXORCISM-4.

It follows from the tables that on small examples, EXORCISM-4 is on average 6 times faster than EXMIN2 and 25 times faster than MINT. On larger examples, for which the experimental results of EXMIN2 and MINT are not available, EXORCISM-4 is 50 times faster than EXORCISM-2 and 2 times faster than EXORCISM-3. In a few cases, EXORCISM-4 was slower than EXORCSIM-3, in particular, for benchmark “apex3.pla” not shown in Table 3.

9 Conclusions

Research described in this paper resulted in the development of a new heuristic ESOP minimizer, EXORCISM-4. EXORCISM-4 compares favorably to other minimizers in terms of minimization quality, is faster and allows us to process larger multi-output Boolean functions. The new tool accepts a wide spectrum of input data formats (PLA, ESOP PLA, BLIF) and unlike EXORCISM-2 and EXORCISM-3, does not require the input to be in the form of a disjoint cover.

The disadvantages of the current implementation are the following: (1) it considers only binary-input data, (2) it does not take don’t-cares into account, and (3) it uses explicit data structures (bit strings) to represent cubes on the cover reshaping stage. The explicit data structures explain why the performance degrades with the increase of the size of the cube cover. However, even functions with covers composed of several thousand cubes can be processed in reasonable time (5-10 minutes).

We developed the new efficient ESOP minimizer hoping that it will serve a variety of applications, and in particular that it will open new opportunities for combining the power of SOP and ESOP minimization to produce efficient implementation of large logic functions.

References

[1] N. Song, M. Perkowski, "EXORCISM-MV-2: Minimization of Exclusive Sum of Product Expressions for Multiple-Valued Input Incompletely Specified Functions," *Proc. ISMVL 1993*, pp. 132-137.

[2] N. Song, M. Perkowski, "Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, Incompletely Specified Functions," *IEEE Trans. on CAD*, Vol. 15, No. 4, April 1996, pp. 385-395.

[3] T. Sasao. "EXMIN2: A Simplified Algorithm for Exclusive-OR-Sum-of Products Expressions for Multiple-Valued-Input Two-Valued-Output Functions". *IEEE Trans. on CAD.* Vol. 12, No. 5, May 1993. pp. 621-632.

[4] T. Kozlowski. *Application of exclusive-OR logic in technology independent logic optimisation.* Ph.D. Thesis. January 1996.

[5] T. Sasao, "An Exact Minimization of AND-EXOR Expressions Using BDDs". *Proc. of IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, 1993*, Germany, pp. 91-98.

[6] T. Sasao, ed., *Representation of Discrete Functions*, Kluwer Academic Publishers, May 1996.

[7] U. Kalay, M. Perkowski, D. Hall, "A Minimal Universal Test Set for Self Test of EXOR-Sum-Of-Products Circuits," *IEEE Trans. Comp.* Vol. 49, #3, March 1999, pp.267-276.

[8] C. Yang, M. Ciesielski, V. Singhal. "BDS: A BDD-based Logic Optimization System". *Proc. of DAC 2000*, pp. 92-97.

[9] A. Mishchenko, B. Steinbach, M. Perkowski. "An algorithm for Bi-Decomposition of Logic Functions". Accepted to *DAC 2001*.

[10] I. I. Zhegalkin, "O tekhnike vychisleniy predlozheniy v simvolicheskoy logike" (About a Technique of Computation of Expressions in Symbolic Logic), *Mat. Sb.* Vol. 34, pp. 9-28, 1927.

[11] I. I. Zhegalkin, "Arifmetizatsiya simvolicheskoy logiki" (Arythmetization of Symbolic Logic), *Mat. Sb.* Vol. 35, pp. 311-377, 1928.

[12] S. M. Reed. "A class of multiple-error-correcting codes and their decoding scheme", *IRE Trans. Information Theory.* Vol. PGIT-4, pp. 38-49. 1954.

[13] D. E. Muller. "Application of Boolean algebra to switching circuit design and to error detection". *IRE Trans. on Electron. Comp.* Vol. EC-3, pp. 6-12, 1954.

[14] D. H. Green, "Families of Reed-Muller canonical forms," *Int. J. Electron.* (UK), vol.70, no.2, 259-280, Feb. 1991.

[15] H. Fleisher, M. Tavel, J. Yeager. "A Computer Algorithm for Minimizing Reed-Muller Canonical Forms". *IEEE Trans. Comp.* Vol. 36. No. 2, pp. 247-250. February 1987.

[16] M.Helliwell, M.A. Perkowski. "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-Muller Forms. *Proc. DAC'88*. pp. 427-432.

[17] J. M. Saul. "An Improved Algorithm for the Minimization of Mixed Polarity Reed-Muller Representations". *Proc. ICCD'90*. pp. 372-375.

[18] T. Kozlowski, E. L. Dagless, J. M. Saul. "An enhanced algorithm for the minimization of exclusive-OR sum of products for incompletely specified functions". *Proc. of ICCD'95*, pp. 244-249.

[19] D. Debnath, T. Sasao. "GRMIN: Heuristic Minimization Algorithm for Generalized Reed-Muller Expressions". *Proc. of IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design 1995*. Japan, pp. 257-264.

[20] R. Drechsler and B. Becker. "Sympathy: Fast Exact Minimization of Fixed Polarity Reed-Muller Expressions for Symmetric Functions." *IEEE Trans. CAD*, Vol. 16, #1, pp. 1-5, Jan. 1997

[21] R. Drechsler, "Pseudo-Kronecker Expressions for Symmetric Functions". *IEEE Trans. Comp.* Vol. 48. No. 8, Sept. 1999, pp. 987-990.

[22] D. Brand, T. Sasao. "Minimization of AND-EXOR expressions using rewriting rules". *IEEE Trans. Comp.* Vol. C42. No. 5, pp. 568-576, May 1993.

[23] A. Zakrevskij. "Minimum Polynomial Implementation of Systems of Incompletely Specified Functions". *Proc. of IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design 1995*. Japan, pp. 250-256.

[24] N. Song. *Minimization of Exclusive Sum of Product Expressions for Multi-Valued Input Incompletely Specified Functions.* M.S. Thesis. EE Dept. Portland State University. Portland, OR, 1992.

[25] N. Song. *A New Design Methodology for Two-Dimensional Logic Cell Arrays.* Ph.D. Thesis. EE Dept. Portland State University. Portland, OR, 1997.

[26] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. *Proc. of DAC '93*, pp. 272-277.

[27] F. Somenzi, *BDD/ADD/ZDD package "CUDD", v.2.3.0*, http://vlsi.colorado.edu/~fabio/

Table 3. Comparison of ESOP minimization results with EXMIN2 [3] and MINT [4]

Benchmark			Cubes			Literals			CPU time, c			Comparison	
Name	Ins	Outs	EXM2	MINT	Exor4	EXM2	MINT	Exor4	EXM2	MINT	Exor4	w/EXM2	w/MINT
5xp1	7	10	34	32	31	186	181	175	13	25	3	0.23	0.12
9sym	9	1	53	51	51	433	427	426	25	52	13	0.52	0.25
add6	12	7	127	127	127	872	936	859	430	320	31	0.01	0.10
addm4	9	8	91	90/89	90/89	654	651	624	129	355	39	0.30	0.11
b12	15	9	28	28	28	164	167	166	4	17	1	0.25	0.06
clip	9	5	68	64	64/63	517	492	479	55	140	11	0.20	0.79
ex7	16	5	81	81	81	601	592	584	46	166	13	0.28	0.08
f51m	8	8	32	31	31	161	185	162	10	22	5	0.50	0.23
in7	26	10	35	35	35	333	352	343	12	56	3	0.25	0.05
intb	15	7	307	267	268	3036	2519	2527	1353	2476	134	0.10	0.05
life	9	1	54	52/51	50/48	415	391	370	23	32	9	0.39	0.28
m181	15	9	29	29	29	169	172	171	5	18	1	0.20	0.06
m4	8	16	84	83	77/76	783	897	714	189	178	120	0.63	0.67
max512	9	6	89	88/83	84/82	696	723	672	71	187	64	0.90	0.34
rd53	5	3	15	16/15	14	60	69	57	2	2	1	0.50	0.50
rd73	7	3	42	36	36	221	194	197	20	36	9	0.45	0.25
rd84	8	4	59	55/54	59/58	330	303	333	45	11	17	0.37	1.54
ryy6	16	1	40	40	40	368	368	368	13	18	2	0.15	0.11
sao2	10	4	29	29/27	28	308	311	288	8	12	1	0.13	0.08
seq	41	35	259	249/248	246	5305	5187	5048	2797	15182	378	0.14	0.02
sym10	10	1	84	82	79	751	735	702	154	176	20	0.13	0.11
t3	12	8	25	25	24	209	214	216	5	10	1	0.20	0.10
t481	16	1	13	13	13	53	53	53	677	377	1	0.01	0.01
vg2	25	8	184	184	184	1992	2033	2010	163	1655	42	0.26	0.03
z4	7	4	29	29	29	145	148	133	4	9	3	0.75	0.33
Total			1891	1804	1791	18762	18300	17677	6253	21532	922	0.15	0.04
Gain			0	-87	-100	0	-462	-1085					
				-4.6%	-5.3%		-2.5%	-5.8%					

Table 4. Comparison of ESOP minimization results with EXORCISM-2 [1] and EXORCISM-3 [2].

Bench mark			Cubes				Literals			Time			Comparison	
Name	Ins	Outs	Ex2	Ex3	Ex4	Ex4b	Ex2	Ex3	Ex4	Ex2	Ex3	Ex4	w/Ex2	w/Ex3
add6	12	7	127	127	127	127	819	800	832	105	24	5	0.05	0.20
alu4	14	8	447	422	435	411	4816	4430	4520	6828	506	140	0.02	0.28
apex1	45	45	285	286	287	285	3796	3820	3998	4697	109	51	0.01	0.48
apex5	117	88	400	399	398	398	4038	4027	4065	20156	457	544	0.03	1.19
cps	24	109	135	135	140	135	2462	2625	3546	467	37	24	0.05	0.64
dule2	22	29	79	78	78	78	920	909	935	72	12	3	0.04	0.25
e64	65	65	65	65	65	65	2210	2272	2270	79	12	1	0.01	0.08
ex5	8	63	72	72	72	71	920	904	975	62	10	5	0.08	0.50
misex3	14	14	545	535	510	501	6837	6632	6141	18897	670	418	0.02	0.62
seq	41	35	245	248	247	246	4833	4822	5063	2996	77	52	0.02	0.68
spla	16	46	260	262	267	259	3420	3393	3969	1223	83	83	0.07	1.00
table3	14	14	166	166	166	166	2491	2491	2630	190	24	17	0.09	0.70
table5	17	15	156	156	156	156	2453	2449	2545	150	33	8	0.05	0.24
vg2	25	8	184	184	184	184	1993	1988	2017	225	29	8	0.04	0.28
Total			3166	3145	3132	3082	43008	41562	43506	59147	2083	1359	0.02	0.51
Gain			0	-21	-34	-84	0	-1446	+498					
				-0.7%	-1.0%	-2.7%		-3.4%	+1.2%					