Brief Contributions

Pseudo-Kronecker Expressions for Symmetric Functions

Rolf Drechsler, Member, IEEE

Abstract—Pseudo-Kronecker Expressions (PSDKROs) are a class of AND/EXOR expressions. In this paper, it is proven that exact minimization of PSDKROs for totally symmetric functions can be performed in polynomial time. A new implementation method for PSDKROs is presented. Experimental results are given to show the efficiency of the presented approach in comparison to previously published work on AND/EXOR minimization.

Index Terms—Logic synthesis, AND/EXOR, PSDKRO, 2-level minimization, BDD.

1 INTRODUCTION

The use of EXOR gates in the synthesis process reduces the hardware costs in many cases [16], [15]. Additionally, EXOR-based circuits often have nice testability properties [8], [11], [12], [4].

In contrast to AND/OR minimization—that, in the meantime, is well-understood—in AND/EXOR minimization, several restricted classes are considered, like Fixed Polarity Reed-Muller Expression (FPRM) [9] and Kronecker Expression (KRO) [5]. (For an excellent overview, see [13].) These subclasses are of interest since the minimization of general Exclusiv Sum of Product Expressions (ESOPs) turned out to be computationally very hard, i.e., all programs presented so far have long runtimes and often fail to determine the optimal result (see, e.g., [14], [7]).

As one alternative, Pseudo-Kronecker Expressions (PSDKROs) [5], [13] have been proposed since they are an interesting compromise: The resulting 2-level forms are of moderate size, i.e., close to ESOPs, and, additionally, the minimization process can be handled within reasonable time bounds.

One subclass of Boolean functions has gained great interest— namely symmetric functions. For totally symmetric functions, it can be proven that ESOPs always allow smaller or equal representation with respect to number of terms than Sum of Product Expressions (SOP) [10]. For the more restricted classes of FPRMs and KROs, exact minimization of totally symmetric functions can be carried out in polynomial time and space [6], [1].

In this paper, it is shown that a similar result can also be proven for PSDKROs. An Ordered Binary Decision Diagram (OBDD) [3] based implementation is proposed that can easily be coded and performs very fast. Experimental results are reported that demonstrate the efficiency of this approach. Totally symmetric functions with more than 20 variables are exactly minimized in less than a hundredth of a second on a Sun Sparc 4 workstation. A comparison to other AND/EXOR classes for some totally symmetric functions and, also, some functions without symmetries is given. Our algorithm is orders of magnitude faster than a hundredth of a second on a Sun Sparc 4 workstation. A new approach based implementation is proposed that can easily be coded and performs very fast. Experimental results are reported that demonstrate the efficiency of this approach. Totally symmetric functions with more than 20 variables are exactly minimized in less than a hundredth of a second on a Sun Sparc 4 workstation.

The decompositions are applied with respect to a fixed variable ordering. Notice that the choice of the variable ordering in which the decompositions are applied and the choice of the decomposition per subfunction largely influences the size of the resulting representation [13].

Example 1. Let \( f(x_1, x_2, x_3) = x_1 + x_2 x_3 \). If we first decompose \( f \) using \( S \), we get:

```c
psdkro (node v) {
  if (v == ZERO) return 0;
  if (v == ONE) return 1;
  if (v.val defined) return v.val;
  v_1 = cofactor.0 (v);
  v_k = cofactor.1 (v);
  v_ex = EXOR (v_1, v_k);
  x1 = psdkro (v_1);
  x2 = psdkro (v_k);
  x3 = psdkro (v_ex);
  v.val = x1 + x2 + x3 - max(x1,x2,x3);
  return v.val;
}
```

Fig. 1. Sketch of the algorithm.

Section 3. The polynomial minimization algorithm is presented in Section 4. The implementation of the algorithm using OBDDs is discussed in Section 5. In Section 6, experimental results are given. We finish with a resume of the results in Section 7.

2 PSEUDO-KRONECKER EXPRESSIONS

In this section, we briefly review the essential definitions of Pseudo-Kronecker Expressions (PSDKROs). (For more details, see [5], [13].)

Let \( f_i (f_0) \) denote the cofactor of \( f \) with respect to \( x_i = 0 (x_i = 1) \) and \( f^i \) is defined as \( f^i := f_0 \oplus f_1 \oplus \cdots \oplus f_i \) being the Exclusive OR operation. A Boolean function \( B^n \rightarrow B \) can then be represented by one of the following formulae:

\[
f = \overline{x}_1 f_0 \oplus x_1 f_1^1 \quad \text{Shannon (S)}
\]

\[
f = f_0 \oplus x f_1 \quad \text{positive Davio (pD)}
\]

\[
f = f_0 \oplus \overline{x}_1 f_1 \quad \text{negative Davio (nD)}.
\]

If we apply to a function \( f \) either \( S \), \( pD \), or \( nD \), we get two subfunctions. To each subfunction again \( S \), \( pD \), or \( nD \) can be applied. This is done until constant functions are reached. If we multiply out the resulting expression, we get a 2-level AND/EXOR form, called a PSDKRO.

The decompositions are applied with respect to a fixed variable ordering. Notice that the choice of the variable ordering in which the decompositions are applied and the choice of the decomposition per subfunction largely influences the size of the resulting representation [13].
3 Symmetric Functions

Let \( f : B^n \rightarrow B \) be a totally defined Boolean function and \( X_n = \{ x_1, \ldots, x_n \} \) be the corresponding set of variables. The function \( f \) is said to be symmetric with respect to a set \( S \subseteq X_n \) if \( f \) remains invariant under all permutations of the variables in \( S \). For completely specified functions, the symmetry is an equivalence relation which partitions the set \( X_n \) into disjoint classes \( S_1, \ldots, S_k \) that will be named the symmetry sets. A function \( f \) is called partially symmetric if it has at least one symmetry set \( S \) with \( |S| > 1 \). If a function \( f \) has only one symmetry set \( S = X_n \), then \( f \) is called totally symmetric. If \( x_i, x_j \in S \subseteq X_n \) (\( x_i \neq x_j, 1 \leq l \leq k \)), \( f \) is called pairwise symmetric in \((x_i, x_j)\). A simple consequence of pairwise symmetry is the following lemma:

**Lemma 1.** A function is pairwise symmetric in \((x_i, x_j)\) iff \( f_{x_i \mapsto 1} = f_{x_j \mapsto 1} \).

A totally symmetric Boolean function can easily be characterized by its value vector, i.e., a Boolean \( n + 1 \)-tuple \([18]\). Since the output value of a totally symmetric function only depends on the number of variables set to 1, \( n + 1 \) bits are sufficient to completely specify the functional behavior.

**Example 2.** The value vector for the EXOR function dependent on 5 variables is given by \((0, 1, 0, 1, 0, 1)\).

4 PSDKROs for Totally Symmetric Functions

In this section, we prove that the minimization of PSDKROs for totally symmetric functions can be performed in polynomial time and space.

Using the decomposition formulae from Section 2, each decomposition of the Boolean function \( f : B^n \rightarrow B \) into two subfunctions \( g, h : B^{n-1} \rightarrow B \) needs two out of the three functions \( f_1, f_2, \) and \( f_3 \). In the following, we show that no more than \( n^3 \) distinct Boolean functions can be obtained out of a totally symmetric Boolean function \( f \) by “repeated applications” of the operations \( f \mapsto f_1, f \mapsto f_2, \) and \( f \mapsto f_3 \). Repeated application in this context means that, starting from the totally symmetric function \( f \), the operators are applied. (It is easy to see that if \( f \) is symmetric, \( f_1, f_2, \) and \( f_3 \) are also symmetric.) Then, the operators are applied again to the newly derived functions. This is iterated until constant functions are obtained.

For a Boolean vector \( a = (a_0, \ldots, a_n) \in \{0, 1\}^{n+1} \) we define the totally symmetric Boolean function \( S[a] : B^n \rightarrow B \) by

\[
S[a](x_1, \ldots, x_n) = a \sum_{l \leq k \leq n} x_l.
\]

(Notice that every totally symmetric Boolean function is uniquely represented by a function \( S[a] \).)

On Boolean vectors, we define three transformations that correspond to \( f_1, f_2, \) and \( f_3 \) on the vector characterization of totally symmetric functions:

\[
(a_0, \ldots, a_n)^{(0)} = (a_0, \ldots, a_{n-1}),
\]

\[
(a_0, \ldots, a_n)^{(1)} = (a_1, \ldots, a_n),
\]

\[
(a_0, \ldots, a_n)^{(2)} = (a_0 \oplus a_1, a_1 \oplus a_2, \ldots, a_{n-1} \oplus a_n).
\]

Then, it directly follows:

**Lemma 2.** For every vector \( a \in \{0, 1\}^{n+1} \), every \( j \in \{0, 1, 2\} \), and every \( i \in \{1, \ldots, n\} \), we have

\[
S[a]_j = S[a^{(j)}].
\]

**Example 3.** Let \( a = (0, 1, 0, 1) \). Then, we get

\[
(0, 1, 0, 1)^{(0)} = (0, 1, 0),
\]

\[
(0, 1, 0, 1)^{(1)} = (1, 0, 1),
\]

\[
(0, 1, 0, 1)^{(2)} = (1, 1, 1).
\]

Now, let \( f = S[a] \) be the totally symmetric Boolean function for which we want to determine the minimal PSDKRO. It is sufficient to prove that no more than \( n^3 \) distinct Boolean vectors can be obtained out of a by repeated applications of the operations \( a \mapsto a^{(0)}, a \mapsto a^{(1)}, \) and \( a \mapsto a^{(2)} \).

For this, we define a function \( \delta \) that describes the mapping of the vectors that results from the applications of the corresponding operations. Starting with a totally symmetric function for each derivation, the corresponding operation can easily be described.

More formally, we obtain:

For \( a = (a_0, \ldots, a_n) \in \{0, 1\}^{n+1} \), \( r, l \in \{0, \ldots, n\} \), and \( C \subseteq \{0, \ldots, n\} \) we define:

1. To keep the definition as simple as possible, the function \( \delta \) is only well defined in the range that is needed later.

**TABLE 1**

<table>
<thead>
<tr>
<th>name</th>
<th>in</th>
<th>FPRM time</th>
<th>KRO time</th>
<th>ESOP time</th>
<th>PSDKRO time</th>
</tr>
</thead>
<tbody>
<tr>
<td>border20</td>
<td>20</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>border25</td>
<td>25</td>
<td>-</td>
<td>14</td>
<td>2</td>
<td>0.1</td>
</tr>
<tr>
<td>co14</td>
<td>14</td>
<td>173</td>
<td>173</td>
<td>52</td>
<td>0.1</td>
</tr>
<tr>
<td>sym0</td>
<td>9</td>
<td>1851.6</td>
<td>0.9</td>
<td>32.7</td>
<td>0.01</td>
</tr>
<tr>
<td>sym10</td>
<td>10</td>
<td>-</td>
<td>266</td>
<td>82</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Implementing the algorithm from [13] can be applied and has polynomial worst case (OBDDs) [3]. The starting point of our algorithm is the OBDD representation of the totally symmetric function that has to be minimized. The size of the representation is $O(n^2)$ due to Lemma 1 [3].

Starting from the root of the OBDD, the graph is recursively traversed and, at each node, an EXOR operation is carried out. An EXOR operation on OBDDs can be performed in polynomial time. Using the fact that, for each of the decomposition formulae from Section 2, two out of the three possible successors $f_0^i, f_1^i,$ and $f^i_2$ are needed, the two minimizing the resulting PSDKRO are chosen. For each node $v$, a minimal number of terms that is needed for the representation as a PSDKRO is stored in the variable $v.val$. Thus, each node has to be evaluated only once. Since, due to Theorem 1, only $n^3$ nodes have to be considered and, at each node, only a polynomial algorithm has to be carried out the complete algorithm has polynomial worst case behavior.

A sketch of the algorithm is given in Fig. 1.

### 6 Experimental Results

In this section, we present experimental results for benchmark functions. All runtimes (if not mentioned otherwise) are given in CPU seconds on a Sun Sparc 4 workstation.

First, we consider some single output functions from [2], [1]. The results, including the runtimes, are given in Table 1. In column FPRM, the results determined by MINT are given. MINT is the exact FPRM minimizer presented in [17]. The runtimes for MINT are given in CPU seconds on a Sequent S27—two 386 processors—machine. (The approach from [17] does not consider symmetry.) In column KRO, the results determined by the exact KRO minimizer for symmetric functions from [1] are given. The runtimes are given in CPU seconds on an HP Apollo 715/50 workstation. The results for the heuristic ESOP minimizer MINT [7] are given in column ESOP. The results of the approach presented in this paper are given in column PSDKRO. As can easily be seen, only the programs who consider symmetry can efficiently minimize the functions. In contrast to the exact KRO minimizer from [1], PSDKROs often allow more efficient representation. Especially on functions of many inputs, the PSDKRO minimizer is much faster (see, e.g., border25). MINT is, in some cases, able to further minimize the number of terms needed for the representation of the function, but, in these cases, it also needs more than 500 times longer. The runtimes of the OBDD-based PSDKRO minimizer for all considered examples are negligible.

In the next series of experiments, we consider multi-output functions that are totally symmetric with respect to each output. The results in comparison with the heuristic ESOP minimizer

### Table 2

<table>
<thead>
<tr>
<th>name</th>
<th>in</th>
<th>out</th>
<th>ESOP terms</th>
<th>ESOP time</th>
<th>PSDKRO terms</th>
<th>PSDKRO time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd53</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>2.1</td>
<td>20</td>
<td>0.01</td>
</tr>
<tr>
<td>rd73</td>
<td>7</td>
<td>3</td>
<td>36</td>
<td>24.1</td>
<td>63</td>
<td>0.01</td>
</tr>
<tr>
<td>rd84</td>
<td>8</td>
<td>4</td>
<td>54</td>
<td>50.4</td>
<td>107</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### Table 3

<table>
<thead>
<tr>
<th>name</th>
<th>in</th>
<th>out</th>
<th>SOP terms</th>
<th>SOP time</th>
<th>ESOP terms</th>
<th>ESOP time</th>
<th>PSDKRO terms</th>
<th>PSDKRO time</th>
</tr>
</thead>
<tbody>
<tr>
<td>add6</td>
<td>12</td>
<td>7</td>
<td>355</td>
<td>229.49</td>
<td>127</td>
<td>332.61</td>
<td>132</td>
<td>0.36</td>
</tr>
<tr>
<td>vg2</td>
<td>25</td>
<td>8</td>
<td>110</td>
<td>932.61</td>
<td>184</td>
<td>324.25</td>
<td>324</td>
<td>2.58</td>
</tr>
<tr>
<td>t481</td>
<td>16</td>
<td>1</td>
<td>481</td>
<td>13</td>
<td>161.49</td>
<td>13</td>
<td>13</td>
<td>0.01</td>
</tr>
<tr>
<td>5xp1</td>
<td>7</td>
<td>10</td>
<td>75</td>
<td>10.97</td>
<td>32</td>
<td>38</td>
<td>38</td>
<td>0.09</td>
</tr>
</tbody>
</table>

The algorithm is implemented in C using Ternary Decision Diagrams (TDDs) [3]. The starting point of our algorithm is the OBDD representation of the totally symmetric function that has to be minimized. The size of the representation is $O(n^2)$ due to Lemma 1 [3].

Starting from the root of the OBDD, the graph is recursively traversed and, at each node, an EXOR operation is carried out. An EXOR operation on OBDDs can be performed in polynomial time. Using the fact that, for each of the decomposition formulae from Section 2, two out of the three possible successors $f_0^i, f_1^i,$ and $f^i_2$ are needed, the two minimizing the resulting PSDKRO are chosen. For each node $v$, a minimal number of terms that is needed for the representation as a PSDKRO is stored in the variable $v.val$. Thus, each node has to be evaluated only once. Since, due to Theorem 1, only $n^3$ nodes have to be considered and, at each node, only a polynomial algorithm has to be carried out the complete algorithm has polynomial worst case behavior.

A sketch of the algorithm is given in Fig. 1.

### Table 2

<table>
<thead>
<tr>
<th>name</th>
<th>in</th>
<th>out</th>
<th>ESOP terms</th>
<th>ESOP time</th>
<th>PSDKRO terms</th>
<th>PSDKRO time</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd53</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>2.1</td>
<td>20</td>
<td>0.01</td>
</tr>
<tr>
<td>rd73</td>
<td>7</td>
<td>3</td>
<td>36</td>
<td>24.1</td>
<td>63</td>
<td>0.01</td>
</tr>
<tr>
<td>rd84</td>
<td>8</td>
<td>4</td>
<td>54</td>
<td>50.4</td>
<td>107</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### Table 3

<table>
<thead>
<tr>
<th>name</th>
<th>in</th>
<th>out</th>
<th>SOP terms</th>
<th>SOP time</th>
<th>ESOP terms</th>
<th>ESOP time</th>
<th>PSDKRO terms</th>
<th>PSDKRO time</th>
</tr>
</thead>
<tbody>
<tr>
<td>add6</td>
<td>12</td>
<td>7</td>
<td>355</td>
<td>229.49</td>
<td>127</td>
<td>332.61</td>
<td>132</td>
<td>0.36</td>
</tr>
<tr>
<td>vg2</td>
<td>25</td>
<td>8</td>
<td>110</td>
<td>932.61</td>
<td>184</td>
<td>324.25</td>
<td>324</td>
<td>2.58</td>
</tr>
<tr>
<td>t481</td>
<td>16</td>
<td>1</td>
<td>481</td>
<td>13</td>
<td>161.49</td>
<td>13</td>
<td>13</td>
<td>0.01</td>
</tr>
<tr>
<td>5xp1</td>
<td>7</td>
<td>10</td>
<td>75</td>
<td>10.97</td>
<td>32</td>
<td>38</td>
<td>38</td>
<td>0.09</td>
</tr>
</tbody>
</table>

The multiple output minimization has been reduced to single output minimization by multivalued interpretation [15].
MINT are given in Table 2. The relations between the two minimizers are the same as for the single output case.

Finally, we compare our minimizer to MINT on a set of (not totally symmetric) benchmark functions. Notice that our minimizer also computes exact solutions in the case of nonsymmetric functions, but the runtime cannot be guaranteed to be polynomial, i.e., the number of OBDD nodes that have to be created might become exponential. The results with respect to runtime and number of terms are given in Table 3. In column SOP, the number of terms for the minimized SOP is given. As can easily be seen, our results can be obtained much faster (up to a factor of 1,000), although our tool was not designed for this case. The results counted in number of terms are only slightly worse than those of the rule-based ESOP minimizer MINT. On the other hand, the results are often much better than the SOP representation (see, e.g., t481).

7 CONCLUSIONS

In this paper, a polynomial algorithm for exact minimization of PSDKROs for totally symmetric functions has been presented. An OBDD-based implementation was given. It performs very fast on all considered symmetric benchmark examples.

With this result, PSDKROs are the largest class of AND/EXOR expressions for which a polynomial algorithm for exact minimization of totally symmetric functions is known.

Finally, we compared our approach to general ESOP minimization. It has been demonstrated that PSDKROs are a good alternative. They are only slightly larger, but can be minimized much faster.

ACKNOWLEDGMENTS

The author would like to thank Ralph Werchner for inspiring the idea of the proof of Theorem 1.