SEPP: a New Compact Three-Level Logic Form

Valentina Ciriani
Department of Information Technologies
Università degli Studi di Milano, Italy
valentina.ciriani@unimi.it

Anna Bernasconi
Department of Computer Science
Università di Pisa, Italy
annab@di.unipi.it

Abstract

Compact area, low delay, and fast synthesis time are important issues in logic circuit design. In order to orchestrate these main goals, this paper proposes a new algebraic three-level expression called sum of extended pseudoproducts (SEPP) with low delay and compact area. These forms are an extension of sum of pseudoproduct (SPP) three-level expressions. The paper proposes an efficient heuristic algorithm that gives compact solutions in a reduced synthesis time. A wide set of experimental results confirms that SEPP forms are often more compact than SPP forms, and their synthesis is always a fast reoptimization phase after SPP minimization.

1 Introduction

The design flow of logic circuits always includes a phase of Boolean function synthesis. In this phase, reduced, and possibly minimal, algebraic forms are determined for the functions, with the aim of simultaneously reducing the final area and the delay of the corresponding circuits. Several logic networks have been proposed and studied, as testified by the vast literature on this subject. The classical approach to logic synthesis is the minimization of two-level Sum of Products (SOP) networks [2]. In this framework the resulting networks have a very low delay, thanks to the fixed number of levels. Although, SOP synthesis can been achieved with very efficient heuristics (e.g., Espresso [2]), SOP circuits are in general not very compact in area.

To build networks with a more compact area, unrestricted multi-level logic synthesis has been widely studied [10]. The drawbacks of this approach are the unbounded number of levels (and therefore the longer delay), as well as the much larger computational time required to synthesize the network. In an attempt to establish an effective trade-off between the two opposite approaches of two-level and multi-level logic, several studies have proposed the optimization of networks with a fixed number of levels (typically, three or four levels). Bounded multi-level logic networks, e.g., EXSOP [4], OR-AND-OR [5], SPP [7], 2SPP [1], have more compact area than SOP forms, and guarantee a still low delay, thanks to their constant and small number of levels. In particular, Sasao [8] statistically showed that three levels of logic are enough to produce a minimal network for most of the Boolean functions; and in many cases three-level logic is a good compromise between circuit speed, circuit size, and the time needed for the minimization procedure. Here we focus on two three-level forms: EXOR-AND-OR and OR-AND-OR forms.

EXOR-AND-OR forms, introduced in [7], are also known as Sum of Pseudoproducts, or SPP. They are a direct generalization of SOP forms, obtained generalizing cubes to pseudocubes where literals in cubes may be replaced by EXOR factors (i.e., exclusive or of literals) in pseudocubes. The idea of introducing EXORs in the synthesized networks comes from the observation that EXOR gates could be sensitive to some structural regularities of Boolean functions that are difficult to express using just AND and OR gates, and that should be exploited in the minimization process in order to derive more compact expressions. To be technologically feasible, SPP networks can be restricted to EXOR-AND-OR forms whose EXOR gates have fan-in bounded by a chosen constant $k$; these forms are called $k$SPP [3]. In this paper we will consider forms with $k = 2$, i.e., 2SPP forms [1].

OR-AND-OR networks [9, 5] are another example of three-level logic networks that are often much more compact than SOP forms. They are defined as the sum (OR) of products (ANDs) of sums (ORs) of literals. More precisely, a product term is either a single literal or a product of
Evaluating 2SEPP forms as an alternative to optimal 2SPPs appears to be a promising approach. These improvements can be obtained in a relatively limited computational time. Therefore, we propose benchmarks that demonstrate the gain in size, which can be quite significant (about 45%). Starting from minimal 2SPP forms, decreases on average, of about 12% when they are transformed into 2SEPPs; and for some cases, the gain can be more significant. An algorithm has been implemented and tested with interesting results (see Section 6), showing how 2SEPP forms are more compact in size than 2SPPs. Indeed, the size of 2SPP networks decreases, on average, of about 12% when they are transformed into 2SEPPs; and for some benchmarks, the gain in size can be quite relevant (about 45%). Starting from minimal 2SPP forms, these improvements can be obtained in really limited computational time. Therefore, evaluating 2SEPP forms as an alternative to optimal 2SPPs appears to be an advisable postprocessing step to be performed after the SPP synthesis in order to achieve very efficiently a more compact bounded level form.

The paper is organized as follows. Sections 2 and 3 recall some preliminary definitions and results on SPP and OR-AND-OR forms, respectively; Section 4 introduces and defines SEPP and 2SEPP forms. Section 5 presents a heuristic method for deriving SEPP and 2SEPP forms starting from SPP and 2SPP forms, respectively. Section 6 discusses the experimental results validating the proposed approach. The paper is concluded in Section 7.

2 \textbf{EXOR-AND-OR Networks}

In this section we briefly recall some basic definitions on SPP and 2SPP networks. SPP networks are three-level EXOR-AND-OR forms introduced in [7] as a direct generalization of SOP forms. They are obtained generalizing cubes to pseudocubes where literals in cubes may be replaced by EXOR factors (i.e., exclusive or of literals) in pseudocubes. The repeated union of pseudocubes yields prime pseudocubes, which extend primes of SOPs; once prime pseudocubes are computed, exact minimization of EXOR-AND-OR forms is reduced to the solution of a covering table, as in case of SOP forms.

Although SPP forms are compact, they are defined for EXOR gates with unbounded fan-in that seem to be impractical [11] in the current technology. It follows that SPP-like forms with a fixed maximum number of literals in the EXOR factors are much more interesting. Therefore 2-SPP forms, where the number of literals in the EXOR factors is at most two, have been proposed. Experimental results show that the 2-SPP synthesis algorithm generates quite compact networks in a short time.

More formally, in a Boolean space \( \{0, 1\}^n \) described by \( n \) variables \( x_1, x_2, \ldots, x_n \), a 2-EXOR factor is an EXOR with at most two variables, one of which possibly complemented. Given two Boolean variables \( x_1, x_2 \), all the possible 2-EXOR factors are essentially \( x_1, \overline{x_1}, x_2, \overline{x_2}, x_1 \oplus x_2 \) and \( x_1 \oplus \overline{x_2} \) (in fact, \( \overline{x_1} \oplus \overline{x_2} = x_1 \oplus \overline{x_2} \), and \( \overline{x_1} \oplus \overline{x_2} = \overline{x_1} \oplus x_2 \)). A 2-pseudoproduct is a product of 2-EXOR factors; a 2-SPP form is a sum of 2-pseudoproducts. A set of points whose characteristic function can be represented as a 2-pseudoproduct is a 2-pseudocube.

For example, the set of points \{0100, 0111, 1100, 1111\} in the Karnaugh map in Figure 1 forms a 2-pseudocube, since its characteristic function is the 2-pseudoproduct \( x_2(x_3 \oplus \overline{x_4}) \). Observe that
this 2-pseudoproduct contains the two products $x_2x_3x_4$ and $x_2\overline{x}_3\overline{x}_4$, grouped by the two dotted lines in the figure. Moreover, note that a product of literals is a particular 2-pseudoproduct containing only 2-EXOR factors formed by single variables. Thus, any cube is a particular 2-pseudocube. Analogously to the SOP forms, the dimension of a 2-pseudocube is the dimension of the space $n$ minus the number of factors in the corresponding 2-pseudoproduct. For example, the 2-pseudocube corresponding to the 2-pseudoproduct $x_2(x_3 \oplus \overline{x}_4)$ in the space $\{0,1\}^4$ has dimension 2. The expression $x_1(x_3 \oplus \overline{x}_1) + x_2(x_3 \oplus \overline{x}_4) + \overline{x}_1\overline{x}_2$ is a 2SPP cover of the function $f$. The corresponding 2SPP circuit representation (i.e., the EXOR-AND-OR network) is shown on the center of the figure.

The exact 2SPP minimization is similar to the Quine-McCluskey method for the SOP synthesis [7]. For example, a minimal 2SPP cover for the function in Figure 1 is $\overline{x}_1\overline{x}_2 + (x_3 \oplus \overline{x}_4)$. Although this exact method for 2SPP minimization performs well on many examples, it is not affordable for all industrial benchmarks. Therefore a heuristic minimization procedure for 2SPP was presented in [1]. This procedure, based on the iteration of a suite of operations that generalize the expansion-irredundant-reduction cycle of heuristic SOP minimization, has been implemented with very good results on industrial benchmarks.

3 OR-AND-OR Networks

This section reviews some basic notions on OR-AND-OR networks [9, 5]. In this context a product term (or product part) is either a single literal or a product of literals provided that a literal of each variable appears at most once, while a sum term is a sum of two or more literals where a literal of each variable appears at most once. A CT (Complex Term) is a conjunction of a product term and sum terms (possibly empty), where two literals corresponding to the same variable are not present both in the product term and in a sum term. The conjunction of the sum terms of a CT is called its sum part. For example, the product $x_1\overline{x}_2(x_3 + x_4 + x_6)(\overline{x}_3 + \overline{x}_4)(\overline{x}_3 + x_5)$ is a CT with the product part $x_1\overline{x}_2$, and the sum part $(x_3 + x_4 + x_6)(\overline{x}_3 + \overline{x}_4)(\overline{x}_3 + x_5)$. A SCT (Sum of Complex Terms) is a disjunction of CTs. For example, $x_1x_2 + x_1\overline{x}_2(x_3 + x_4 + x_6) + (x_3 + \overline{x}_4)(\overline{x}_3 + x_5)$ is a SCT containing 3 CTs.

The minimization heuristic proposed in [5] is based on three operators: algebraic method, Boolean method and merging method. The algebraic method produces a CT combining a set of CTs and is based on classical algebraic methods used in multilevel synthesis [6]. The Boolean method combines two CTs in a single CT generating only OR-AND expressions, and it is a subset of the Boolean operators used in multilevel synthesis [6]. Finally the merging method merges two CTs in a single CT by combining their sum parts. The overall synthesis technique consists in generating all the possible CTs using the three methods, and in successively selecting a subset of them that covers the given function. The experimental results show that this is a fast synthesis technique and that the generated OR-AND-OR forms are very compact.

4 SEPP Algebraic Forms

Mirroring the definitions of (2-)EXOR-AND-OR and OR-AND-OR networks we now merge them in an ((2-)EXOR/OR)-AND-OR network whose algebraic forms are called SEPP (Sum of Extended PseudoProducts) and 2SEPP.
An EXOR term is an exclusive sum of two or more literals where a literal of each variable appears at most once, in particular a 2-EXOR term is an EXOR term of exactly two literals.

Observe that an EXOR term is equivalent to a conjunction of sum terms (e.g., $(x_1 \oplus x_2) = (x_1 + x_2)(\overline{x}_1 + \overline{x}_2)$). Thus when a sum term is implied by an EXOR term, their product is equivalent to the EXOR term alone, for instance, since $(x_1 \oplus x_2)$ implies $(\overline{x}_1 + \overline{x}_2)$ then $(x_1 \oplus x_2)(\overline{x}_1 + \overline{x}_2) = (x_1 \oplus x_2)$. Therefore, we can define an ECT as follows.

**Definition 1** An ECT (Extended Complex Term) is a conjunction of a product term, sum terms, and EXOR terms (possibly empty), where two literals corresponding to the same variable are not present both in the product term and in a sum or EXOR term, and where each EXOR term does not imply any of the sum terms. The conjunction of the EXOR terms of an ECT is called its EXOR part.

Finally we can define the SEPP and 2SEPP expressions.

**Definition 2** A SEPP form is a sum of ECTs and a 2SEPP is a sum of ECTs whose EXOR part contains 2-EXOR terms only.

For example, $x_1\overline{x}_5(x_2 \oplus x_3)(x_3 + x_4 + x_6) + x_1x_2(x_3 + x_5) + (x_1 \oplus x_4)(x_1 \oplus x_3 \oplus x_5) + x_4$ is a SEPP expression. Note that since a SOP is a special case of a SEPP (2SEPP) expression, each Boolean function can be expressed in SEPP (2SEPP) form.

Extending the definition of irredundancy to SEPP, we say that a SEPP (2SEPP) form $S$ is irredundant when removing an ECT from $S$ represents a different function. The networks corresponding to SEPP (2SEPP) expressions are three-level logic networks containing the first layer of EXOR and OR gates, the second layer containing AND gates and a final OR gate. For example, the 2SEPP form $(x_1 + x_2)(x_3 \oplus x_4) + \overline{x}_2\overline{x}_3$ is represented on the right side of Figure 1.

## 5 Synthesis of SEPP Forms

In this section we propose a heuristic method for deriving SEPP and 2SEPP forms.

The algorithm (described in Figure 2) starts with the SPP (or 2SPP) minimization of the given function $f$ (using the algorithms [3] for SPP minimization or [1] for the 2SPP synthesis). The second step of the algorithm decomposes the EXOR terms of the SPP (2SPP) form in a product of sum terms. Note that now the expression is represented as a sum of CTs. The third phase iteratively replaces couples of CTs with singles CT equivalent to their sum, following standard algebraic and Boolean operations [5] described in the following. This step ends when no more couples of CTs can be further merged. The last phase reconstructs the EXOR terms in the CTs. After this step the algorithm stops producing a SEPP form. For simplicity, hereafter we will address only the 2SEPP problem, since the SEPP minimization is a simple generalization.

We now describe the logic operators used by the algorithm. After the 2SPP synthesis the algorithm transforms the EXOR terms in products of sum terms. In the context of 2SPP forms, this is simply due to the equivalence $(h \oplus l) = (h + l)(\overline{h} + \overline{l})$ where $h$ and $l$ are literals.
Let \( p \) be a CT and \( h, l \) be two literals. The algebraic method considers two CTs that differ in a literal only, i.e., \( p \cdot h \) and \( p \cdot l \), that are removed from the sum of CTs and replaced with \( p \cdot (h + l) \). This directly follows from the rule: \( p \cdot h + p \cdot l = p \cdot (h + l) \). For example, the two CTs \( x_1x_2(x_3 + \overline{x}_3) \) and \( x_1x_3(x_2 + \overline{x}_2) \) are replaced by \( x_1(x_2 + x_3) \).

The Boolean method is based on two rules of the Boolean algebra. Let \( a, b \) and \( c \) be sum terms or single literals. The first Boolean rule we use is \( p \cdot a + p \cdot b \cdot c = p \cdot (a + b) \cdot (a + c) \). Thus, the CTs \( p \cdot a \) and \( p \cdot b \cdot c \) are replaced with \( p \cdot (a + b) \cdot (a + c) \). Letting \( h, l \), and \( m \) be literals, the second rule is \( p \cdot h \cdot l + p \cdot \overline{h} \cdot m = p \cdot (h + m) \cdot (\overline{h} + l) \). Thus, the algorithm replaces the CTs \( p \cdot h \cdot l \) and \( p \cdot \overline{h} \cdot m \) with the CT \( p \cdot (h + m) \cdot (\overline{h} + l) \).

Finally, the algorithm uses the merging rule proposed in [5]. Let \( s \) and \( t \) be two different sum terms. Consider two CTs differing in one sum term only, i.e., \( p \cdot s \) and \( p \cdot t \); we can replace their sum with \( p \cdot (s + t) \), where we can further simplify the sum term \((s + t)\) when \( s \) and \( t \) have common literals.

In summary, after the 2SPP minimization we apply OR-AND-OR operators in order to merge pseudoproducts and insert sum terms in the form. Observe that the naive solution that does not consider EXOR terms, and tries to construct CTs starting only from the product terms in the pseudoproduct would give worst results. Indeed, this solution is simple, but does not take into account the EXOR terms that could be useful in deriving a small SEPP form. For example, consider the 2SPP form \((x_1 \oplus x_2)x_3 + x_1x_3x_4 + x_2x_3x_4\), suppose not to expand the EXOR term, thus we can only use the algebraic method on \( x_1x_3x_4 + x_2x_3x_4 \) obtaining the 2SEPP: \((x_1 \oplus x_2)x_3 + (x_1 + x_2)x_3x_4\). If we expand the EXOR term we have \((x_1 + x_2)(\overline{x}_1 + \overline{x}_2)x_3x_4\) and applying the merging method we obtain the smaller 2SEPP expression \((x_1 + x_2)(\overline{x}_1 + \overline{x}_2 + x_3)x_4\).

The SEPP expression derived by the proposed algorithm is correct and irredundant, as shown in the following proposition.

Proposition 1: Given a function \( f \), the SEPP form computed by the algorithm in Figure 2 covers \( f \) and is irredundant.

Proof. (Sketch) Since we start from a minimized SPP (or 2SPP) form that covers \( f \), and we replace couples of CTs with an equivalent CT, the resulting SEPP form covers the given function \( f \). Moreover, the algorithm for SEPP and 2SEPP synthesis returns an irredundant form. In fact, note that each extended pseudoproduct of the resulting SEPP corresponds to a pseudoproduct or to a sum of more pseudoproducts of the original SPP (2SPP) form. Since the SPP and 2SPP algorithms return an irredundant expression, deleting an extended pseudoproduct from the SEPP form changes the covered function.

6 Experiment Results

In this section we discuss the computational results obtained by building 2SEPP expressions for a given Boolean function \( f \). We have considered the well known Espresso benchmark suite [12] and the computational experiments were performed on a Pentium 1.6 GHz PC with 1 GB of RAM.

Table 1 reports a cost-oriented comparison among the starting 2SPP forms and the 2SEPP expressions produced by the algorithm: the first column reports the name of the instance, the following pair of columns report the number of literals of the 2SPP and 2SEPP forms, the forth column shows the gain of each benchmark, and the last column reports the computational time in seconds for deriving the 2SEPP from the starting 2SPP. Due to the limited space available, we report only a significant subset of the experiments.

As Table 1 shows, the time overhead added by 2SEPP construction is quite limited. On average, each 2SEPP form requires around 0.01 seconds. On the instances reported in Table 1, the 2SEPP form is the cheapest and often exhibits remarkable improvements with respect to the 2SPP form: see the 45% cost reduction achieved on instance \( 6481 \), and the 41% improvement achieved on instance \( ry96 \). On average, 2SPP forms are more compact then 2SPPs of about 12%. Given that the time required to obtain this potential improvement is rather limited, the evaluation of 2SEPP forms appears to be an advisable post-processing strategy to obtain a more compact expression starting from a 2SPP form.
Table 1: Cost and computational times (in seconds) of the 2SEPP starting from a 2SPP form.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>2SPP Cost</th>
<th>2SEPP Cost</th>
<th>Gain</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>addm4</td>
<td>1460</td>
<td>1373</td>
<td>6%</td>
<td>0.001</td>
</tr>
<tr>
<td>al2</td>
<td>237</td>
<td>130</td>
<td>46%</td>
<td>0.001</td>
</tr>
<tr>
<td>b7</td>
<td>106</td>
<td>104</td>
<td>2%</td>
<td>0.001</td>
</tr>
<tr>
<td>br1</td>
<td>450</td>
<td>444</td>
<td>1%</td>
<td>0.001</td>
</tr>
<tr>
<td>chkn</td>
<td>1440</td>
<td>1137</td>
<td>21%</td>
<td>0.001</td>
</tr>
<tr>
<td>cor</td>
<td>719</td>
<td>666</td>
<td>8%</td>
<td>0.001</td>
</tr>
<tr>
<td>dist</td>
<td>951</td>
<td>903</td>
<td>5%</td>
<td>0.001</td>
</tr>
<tr>
<td>ex7</td>
<td>635</td>
<td>504</td>
<td>21%</td>
<td>0.001</td>
</tr>
<tr>
<td>exps</td>
<td>2180</td>
<td>2101</td>
<td>4%</td>
<td>0.001</td>
</tr>
<tr>
<td>fmi</td>
<td>524</td>
<td>467</td>
<td>11%</td>
<td>0.001</td>
</tr>
<tr>
<td>in0</td>
<td>1194</td>
<td>1312</td>
<td>12%</td>
<td>0.001</td>
</tr>
<tr>
<td>in5</td>
<td>1260</td>
<td>957</td>
<td>24%</td>
<td>0.001</td>
</tr>
<tr>
<td>int</td>
<td>420</td>
<td>360</td>
<td>14%</td>
<td>0.001</td>
</tr>
<tr>
<td>n4</td>
<td>1520</td>
<td>1254</td>
<td>5%</td>
<td>0.001</td>
</tr>
<tr>
<td>mipl4</td>
<td>1024</td>
<td>960</td>
<td>6%</td>
<td>0.001</td>
</tr>
<tr>
<td>opa</td>
<td>1104</td>
<td>950</td>
<td>14%</td>
<td>0.001</td>
</tr>
<tr>
<td>pdc</td>
<td>1207</td>
<td>1151</td>
<td>5%</td>
<td>0.001</td>
</tr>
<tr>
<td>prom2</td>
<td>5202</td>
<td>5088</td>
<td>2%</td>
<td>0.001</td>
</tr>
<tr>
<td>root</td>
<td>406</td>
<td>389</td>
<td>5%</td>
<td>0.001</td>
</tr>
<tr>
<td>ryy6</td>
<td>624</td>
<td>368</td>
<td>41%</td>
<td>0.001</td>
</tr>
<tr>
<td>spia</td>
<td>2764</td>
<td>2937</td>
<td>3%</td>
<td>0.001</td>
</tr>
<tr>
<td>sym10</td>
<td>881</td>
<td>838</td>
<td>5%</td>
<td>0.001</td>
</tr>
<tr>
<td>t1</td>
<td>357</td>
<td>278</td>
<td>22%</td>
<td>0.001</td>
</tr>
<tr>
<td>t481</td>
<td>4792</td>
<td>2629</td>
<td>45%</td>
<td>0.375</td>
</tr>
<tr>
<td>x1dn</td>
<td>491</td>
<td>397</td>
<td>20%</td>
<td>0.015</td>
</tr>
</tbody>
</table>

7 Conclusion

This paper presented a new three-level logic form with the first level containing OR and EXOR gates combined by AND gates, which are summed in the final OR gate.

A possible future study is to perform a post-processing phase after OR-AND-OR synthesis using 2SPP operators [1] in order to find a SEPP form, and to compare the obtained SEPP forms with the ones derived from 2SPP networks and OR-AND-OR manipulations described in this paper. An alternative approach is to define a heuristic for the direct synthesis of SEPP networks using simultaneously the minimization operators proposed in literature for OR-AND-OR and EXOR-AND-OR synthesis.

References