SNF: A Special Normal Form for ESOPs

Bernd Steinbach
Freiberg University of Mining and Technology
Institute of Computer Science
D-09596 Freiberg, Germany
steinb@informatik.tu-freiberg.de

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

Abstract

This paper introduces a new normal form for Exclusive Sums-of-Products (ESOPs) of completely specified Boolean functions. We study the properties of the SNF and show its special place among canonical Reed-Muller representations. We propose to use the SNF in a number of applications related to the exact ESOP minimization. We describe an efficient way to compute the SNF with the complexity proportional to the number of nodes in the BDD of the given function. Experimental results speak for the potential usefulness of the SNF.

1 Introduction

The problem of finding an Exclusive Sum-Of-Product (ESOP) of the given Boolean function with the minimum number of cubes (and the minimum number of literals, as a secondary goal) has both theoretical and practical value.

From the theoretical point of view, ESOP minimization is interesting because ESOP is the most general Reed-Muller forms with many interesting properties. From the practical point of view, EXOR gates and ESOPs have numerous applications in logic synthesis and design-for-test. In particular, it has been shown [11] that the ESOP representation of Boolean functions is typically more compact that the SOP representation. For some functions (e.g. the parity function), the number of cubes in ESOP is linear in the number of variables while the number of cubes in SOP is exponential.

Research in exact ESOP minimization has a long history. Only the most important contributions are mentioned here.

In 1988, the first systematic approach has been introduced [1][3] reducing the problem of exact ESOP minimization to that of finding a satisfying assignment of the constraint (called Helliwell function). However, the practical value of this approach is limited, because the number of variables in Helliwell function is equal to the number of cubes that can appear in the minimum ESOP (for the most functions of n variables, this number is 3^n).

In 1993, an extension of the minimization procedure based on Helliwell function has been proposed [10][3]. This approach shows how to minimize a function of n variables if the exact minimum for all functions of n-k (k≥1) variables is known. As a result of problem decomposition, a specialized constraint is generated, which is similar to Helliwell function but depends on fewer variables. The constraint is represented using BDDs [10] or ZDDs [8] and the solution is found as a shortest path in the decision diagram. BDD reduction techniques are used to reduce the size of the constraint by incrementally eliminating those variable assignments that lead to solutions above an upper bound on the number of cubes [3].

The method presented in [10] is the only one to find exact minimum for *arbitrary* function of five input variables. This method can also minimize some function up to ten variables, as long as the number of variables in the constraint can be limited using a tighter upper bound on the number of cubes in the exact minimum ESOP.

Other approaches to exact minimization target not the general class of functions, but functions that satisfy certain properties. For example, in [1] satisfiability is used to solve the minimization problem for functions whose minimum ESOP is known to contain the given number of cubes.

The minimization procedure proposed in this paper essentially differs from the previous ones. The problem is solved by first deriving a canonical normal form, called the SNF, and next transforming it into the exact minimum ESOP (or the set of exact minimum ESOPs).

The SNF is also useful to prove exactness of ESOPs found using other methods, because it gives a lower bound on the number of cubes in the minimum solution. Finally, the SNF provides valuable characterization of the given function, which cannot be found using other methods.

The rest of the paper is organized as follows. Section 2 gives the basic definitions. Section 3 introduces the SNF and presents a naïve way of its computation. Section 4 discusses the properties of SNF and proves the canonicity. Section 5 shows applications of SNF. Section 6 gives an efficient algorithm for SNF computation from the BDD of the given function. Section 7 contains experimental results. Section 8 concludes the paper.

2 Preliminaries

This section introduces definitions and basic knowledge used in the paper.

A *literal* is a Boolean variable in negative or positive polarity. A *cube C* 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 D* between two cubes is the number of variables, in which the cubes differ.

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

For example, assuming that cube $a\bar{b}\bar{d}$ belongs to a 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.

The Exclusive-OR (EXOR) operation $\oplus$ is a *linear* operation and is defined in Table 1.

Table 1. Definition of the EXOR operation

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

The *fundamental property* of Exclusive-OR (EXOR) operation is the following:

$$x \oplus \bar{x} \oplus 1 = 0. \tag{1}$$

The operation of addition in this paper is used in the sense of Exclusive-OR. For example, "adding this pair of cubes to the function *f*" means creating an Exclusive-OR of the two cubes and the function *f* represented as an ESOP.

An *Exclusive-Or Sum-Of-Products* (*ESOP*) is an Exclusive-OR of zero or more cubes.

Proposition 1. Two identical cubes can be added to any ESOP without changing the function represented by it.

This statement is clear from the following formulas:

$$f = f \oplus 0 \tag{2}$$

$$0 = C \oplus C \tag{3}$$

$$f = f \oplus C \oplus C \tag{4}$$

The number of ESOP's for each Boolean function is *infinite*, because Proposition 1 can be applied many times.

Using Proposition 1 in the reverse direction, to collapse all the pairs of the identical cubes, leads to an ESOP that is *unique* for the class of ESOPs generated by Proposition 1.

An ESOP is *reduced* if it does not contain identical cubes. The number of all cubes of *n* variables is 3^n and consequently there are 2^{3^n} reduced ESOPs. The number of reduced ESOPs of each of the 2^{2^n} Boolean function of *n* variables is *finite*. This paper focuses on reduced ESOPs.

An ESOP is *minimum* (or, *exact minimum*) if it contains the minimum number of cubes among all the ESOPs of the given Boolean function.

Two ESOPs are considered *different* if they differ in at least one cube, that is, if at least one of the ESOPs has a cube, which the other ESOP does not have.

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

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

A functional representation is *canonical* if for any function there exists only one representation of this kind.

The *minterm* is a cube containing all the literals of the support. The *minterm ESOP* of a function is the ESOP created by adding the cubes representing all the minterms of the given function. By definition of canonicity, the minterm ESOP is a canonical representation, because the set of minterms is a unique characteristic of the function.

Some helpful concepts can be defined using the distances between the cubes of an ESOP.

Adjacency Graph *AG*(*V*, *E*) of a reduced ESOP

Definition. The vertices *V* of the adjacency graph *AG*(*V*,*E*) correspond to the cubes of the reduced ESOP. Two vertices *V* of *AG*(*V*, *E*) are connected by an edge, if the associated cubes have distance 1.

It is possible to represent an ESOP by a list of ternary vectors [12]. Each cube of the ESOP is represented by a ternary vector having the length of the support of the ESOP. The variables of the cube are encoded as follows:

0	if the positive literal belongs to the cube
1	if the negative literal belongs to the cube
-	if the variable does not appear in the cube.

For example, the function $f(a,b,c,d) = a\bar{b}\bar{d} \oplus b\bar{c}$ may be represented by two ternary vectors listed in the table:

$$f = \begin{array}{cccc} a & b & c & d \\ \hline 1 & 0 & - & 0 \\ - & 1 & 0 & - \end{array}$$

Assuming the above encoding, the AND-operation between the ternary elements of a row and the EXOR-operation between the ternary vectors the ESOP expression of the function can be easily performed using the ternary vector list (TVL).

3 Definition of SNF

This section introduces a new canonical Reed-Muller expansion for completely specified Boolean functions. The proof of canonicity is given in the next section.

The fundamental property of Exclusive-OR (1) can be expressed in any of the following forms:

$$x = \bar{x} \oplus 1 \tag{5}$$

$$\bar{x} = 1 \oplus x \tag{6}$$

$$1 = x \oplus \bar{x} \tag{7}$$

These three formulas show that each element of the set $\{x, \bar{x}, 1\}$ has isomorphic properties. For each variable in the support of the Boolean function *f*, exactly one left-hand-side element of (5), (6) or (7) is included in each cube of an ESOP of function *f*.

Using the appropriate formula, the cube *C* can be *expanded with respect to* (*w.r.t.*) *variable x* into two cubes having all the variables the same as in the original cube, except variable *x*. This variable appears in two cubes in the forms that are complementary w.r.t. the form it takes in the original cube (5), (6), (7). This expansion is performed by the function expand, which takes the cube and the variable and returns the two resulting cubes.

Suppose the support of a Boolean function is (*a*, *b*, *c*, *d*) and $a\bar{b}\bar{d}$ is one of the cubes in the ESOP. There exist four different expansions w.r.t. each variable of the support.

expansions w.r.t. *a*: $a\bar{b}\bar{d} = (\bar{a} \oplus 1) \cdot \bar{b}\,\bar{d} = \bar{a}\,\bar{b}\,\bar{d} \oplus \bar{b}\,\bar{d}$

expansions w.r.t. *b*: $a\bar{b}\bar{d} = a \cdot (1 \oplus b) \cdot \bar{d} = a\bar{d} \oplus ab\bar{d}$

expansions w.r.t. *c*: $a\bar{b}\bar{d} = a\bar{b} \cdot (c \oplus \bar{c}) \cdot \bar{d} = a\bar{b}c\bar{d} \oplus a\bar{b}\,\bar{c}\,\bar{d}$

expansions w.r.t. *d*: $a\bar{b}\bar{d} = a\bar{b} \cdot (1 \oplus d) = a\bar{b} \oplus a\bar{b}d$

It is obvious that the right hand side ESOPs of the above four examples represent the same function $a\bar{b}\bar{d}$.

The expansion of a cube w.r.t. one variable can be repeated for all variables of the support. The result does not depend on the order of the *n* expanded variables and is an ESOP consisting of exactly 2^n cubes. It can be shown that the set of 2^n cubes resulting from expanding the given cube forms a complete lattice. It is possible to define operations, join and meet, for the elements of this lattice. We do not introduce these operations because they are not necessary for our presentation in this paper.

The lattice of cubes created by expanding cube $a\bar{c}$ with the support (*a*, *b*, *c*) is shown in Figure 1. Notice that the cubes connected by lines differ in exactly one variable. For each cube in the lattice the number of cubes connected by the lines is equal to the size of the support *n* (in the example of Figure 1, the size of the support is 3).

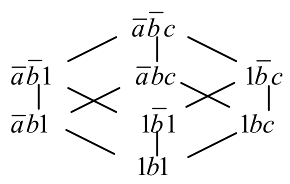


Figure 1. The cube lattice created by expanding cube $a\bar{c}$.

Algorithm Exp(*f*)

Given: any ESOP of a Boolean function *f*
Result: complete expansion of the Boolean function *f* w.t.r. all variables of its support

```
for all variables Vi of the support
   for all cubes Cj of f
   { (Cn1, Cn2) = expand(Cj , Vi)
     replace Cj by (Cn1, Cn2)
   }
```

Suppose the given ESOP of a Boolean function *f* with a support of *n* variables consist of *k* cubes. The result of Exp(*f*) is an ESOP, which includes $k \cdot 2^n$ cubes.

If more than one cube belongs to *f* , Exp(*f*) contains pairs of identical cubes. A reduced ESOP of the same function can be derived by Algorithm R.

Algorithm R(*f*)

Given: any ESOP of a Boolean function *f* containing *k* cubes
Result: reduced ESOP of *f*

```
for i = 0 to k - 2
   for j = i + 1 to k - 1
   { if (C[i] == C[j])
      { C[i] = C[k-1]
        C[j] = C[k-2]
        k = k - 2
        j = i
      }
   }
```

Using the algorithms Exp(*f*) and R(*f*) it is possible to create a special ESOP having a number of remarkable properties.

Definition of SNF(*f*)

Take any ESOP of a Boolean function *f*. The resulting ESOP of

SNF(*f*) = R(Exp(*f*))

is called *Specialized Normal Form* (SNF) of the Boolean function.

Note that the definition of the SNF(*f*) involves two steps. First, the complete expansion of all cubes of *f* w.r.t. all the variables. Second, all pairs of identical cubes are removed in order to obtain an reduced ESOP of *f*. For practical reasons, these two processes (expansion and reduction) may be carried out simultaneously.

Consider function NEXOR of two variables, *a* and *b*, and one of its ESOPs: ($a \oplus \bar{b}$). To compute the SNF (Figure 1), first create the expansion of cube a, then create the expansion of cube $\bar{b}$, and finally find the union of the resulting cubes while eliminating the duplicated cubes (marked by the cross in Figure 1). The ones in the table stand for don't-care literals in the cubes and may removed.

a	$\bar{b}$	SNF($a \oplus \bar{b}$)
× $\bar{a}b$	× $\bar{a}b$	$\bar{a}\bar{b}$
$\bar{a}\bar{b}$	$\bar{a}\cdot 1$	$1\cdot b$
$1\cdot b$	ab	$1\cdot \bar{b}$
$1\cdot \bar{b}$	$a\cdot 1$	$\bar{a}\cdot 1$
		ab
		$a\cdot 1$

Figure 2. Example of SNF computation.

4 Properties of SNF

To prove that the SNF is a canonical representation, two transformations are introduced and the following two lemmas should be proved.

Transformation T1: Adding a pair of duplicated cubes.

Transformation T2: Combining two distance-1 cubes.

Lemma 1. Any ESOP can be created from the minterm ESOP by repeatedly applying transformations T1 and T2.

Proof. Take an ESOP and break it down into elementary cubes representing minterms. In each vertex of the Boolean space, count the number of elementary cubes that cover this vertex. Observe that those vertices, where the function is one (zero), are covered by an odd (even) number of elementary cubes (according to Proposition 1).

Now consider the minterm ESOP. This ESOP covers each vertex of the Boolean space where the function is one with a single elementary cube. Add the even number of cubes (transformation T1) to each vertex of the Boolean space according to the counting performed above. Combine smaller cubes into larger cubes (transformation T2) until the cubes of the original ESOP are created. Q.E.D.

Lemma 2. Transformations T1 and T2 do not change the SNF of the function *f*.

Proof. Consider transformation T1. After adding two identical cubes to the original ESOP of *f*, Exp(f) contains two sets of identical expanded cubes. After applying reduction R(Exp(f)), the cubes in the identical sets are reduced and SNF(f) remains unchanged.

Consider transformation T2. The expansions of the two distance-1 cubes are similar, except for the single variable, in which they differ. Exactly half of the cubes in each expansion are the same. As a result of adding these two sets of cubes, the identical cubes will be reduced, while the remaining cubes will add up to the expansion of the cube created by merging the distance-1 cubes. Q.E.D.

The second part of this proof is illustrated by the following example: $a\bar{b}c \oplus a\bar{b}\bar{c} = a\bar{b}$. Expansions of cubes $a\bar{b}c$ = (101), $a\bar{b}\bar{c}$ = (100), and $a\bar{b}$ = (10-) are given in Figure 3.

It is easy to see that adding the expansions of the first two cubes and removing the duplicated cubes (marked by "×" in Figure 3) leads to the expansion of the third cube.

	abc			abc			abc
	010			011			010
	01-	×		01-	×		011
	0-0			0-1			0-0
$a\bar{b}c =$	0--	×	$a\bar{b}\bar{c} =$	0--	×	$a\bar{b} =$	0-1
	-10			-11			-10
	-1-	×		-1-	×		-11
	--0			--1			--0
	---	×		---	×		--1

Figure 3. An example illustrating Lemma 2.

Theorem 1. The SNF(f) is canonical.

Proof. Take two arbitrary ESOPs of the same function. According to Lemma 1, each of them can be derived from the minterm ESOP using transformations T1 and T2. Because transformations T1 and T2 do not change the SNF (Lemma 2), SNFs for these ESOPs are the same and equal to SNF of the minterm ESOP. Q.E.D.

The following lemmas and theorems elaborate on the properties of the SNF and show how it can be used to prove a number of statements about the set of all ESOPs and the set of exact minimum ESOPs of the Boolean function.

Lemma 3. Every reduced ESOP of the Boolean function *f* can be derived by adding the SNF(f) to a reduced ESOP of the constant zero function and removing the duplicated cubes.

Proof. Consider one ESOP, *W*, of the function *f*. *W* may have some cubes in common with SNF(f) and differ from SNF(f) in all the other cubes. Let us construct an ESOP *Z* by adding all the cubes from SNF(f) that do not appear in *W* and all the cubes in *W* that do not appear in SNF(f).

By construction of Z, SNF(f) = $W \oplus Z$. Consequently, Z = SNF(f) $\oplus$ W. Because SNF(f) and W are ESOPs of the same function, and because the duplicated cubes have been removed while adding them, Z is a reduced ESOP of the constant zero function. Because no assumptions are about W, this method constructs the required ESOP of the zero function for any given ESOP. Q.E.D.

Lemma 4. Adding SNF(f) to two different reduced ESOPs of the constant zero function and removing the duplicated cubes leads to two different reduced ESOPs.

Proof. Assume that the statement of the theorem is wrong and there are two different reduced ESOPs of the zero function, Z_1 and Z_2, which when added to SNF(f) lead to two identical reduced ESOPs.

From the fact that Z_1 and Z_2 are two different ESOPs, we conclude that there is at least one cube in one of them that is not present in the other. Without limiting the generality of the following statements, assume that Z_1 has cube C that does not appear in Z_2.

Consider two cases: (1) cube C appears in SNF(f) and (2) cube C does not appear in SNF(f). As a result of removing duplicated cubes, in case of (1), cube C will not appear in SNF(f) $\oplus Z_1$ but will appear in SNF(f) $\oplus Z_2$. In case of (2), cube C will appear in SNF(f) $\oplus Z_1$ but will not appear in SNF(f) $\oplus Z_2$. In both cases, the resulting ESOPs, SNF(f) $\oplus Z_1$ and SNF(f) $\oplus Z_2$, are different. This is a contradiction. Q.E.D.

Theorem 2. Every Boolean function has exactly $2^{(3^n)-(2^n)}$ different reduced ESOPs.

Proof. From the above two lemmas, it follows that for any Boolean function, it is possible to derive exactly as many different reduced ESOPs as there exist different reduced ESOPs of the constant zero function. It means that the number of different reduced ESOPs is the same for all Boolean functions. One ESOP cannot represent two different functions. Because there are $2^{(3^n)}$ different reduced ESOPs and $2^{(2^n)}$ different functions, each function has exactly $2^{(3^n)-(2^n)}$ different reduced ESOPs. Q.E.D.

Theorem 3. (Weak lower bound on the size of the minimum ESOP). No ESOP representation of a Boolean function f contains less than $\lceil \,|\,\mathrm{SNF}(f)\,|\,/\,2^n\, \rceil$ cubes (the smallest integer number greater than the number of cubes in the SNF(f) divided by 2^n).

Proof. Assume that the theorem is wrong. Suppose there exists an ESOP M of the given function f containing less than $k = \lceil\, |\,\mathrm{SNF}(f)\,|\,/\,2^n\, \rceil$ cubes, $|\,M\,| < k$. Exp(M) contains $|\,M\,| * 2^n$ cubes and the final R-operation removes some cube to create SNF(M) = R(Exp(M)). Thus,

$$|\,\mathrm{SNF}(M)\,| < |\,M\,| * 2^n < k * 2^n \tag{8}$$

$$|\,\mathrm{SNF}(M)\,| / 2^n < |\,M\,| < k \tag{9}$$

$$\lceil |\,\mathrm{SNF}(M)\,| / 2^n \rceil \le |\,M\,| < k = \lceil |\,\mathrm{SNF}(f)\,| / 2^n \rceil \tag{10}$$

and the SNF(M) is different from SNF(f). According to Theorem 1 is SNF a canonical representation and the two different SNFs describe two different functions M and f. This is a contradiction, which proves the theorem. Q.E.D.

In fact, it is possible to formulate a stronger lower bound on the number of cubes in the exact minimum ESOP by considering the cubes appearing in the SNF. This will be shown in the next section.

The following theorem justifies the exact ESOP minimization algorithm described below.

Theorem 4. Adding the smallest number of pairs of the identical cubes to SNF(f) in such a way that the complete expansions of cubes are created, leads to the exact minimum ESOP of the given function.

Proof. According to the Theorem 2, there are $2^{(3^n)-(2^n)}$ different ESOPs of the function f. This finite set of ESOPs contains one or more exact minimum ESOPs including k_{min} cubes and other ESOPs including k_i' cubes, $k_{min} < k_i$' , as well. Without limiting the generality of the following statements, assume that M is an exact minimum ESOPs of the function f , $|\,M\,| = k_{min}$ and L is an ESOP of the same function f including k_L' cubes, $k_{min} < k_L$'. After expansions of M and L holds

$$\mathrm{Exp}(M) = k_{min} * 2^n < k_L' * 2^n = \mathrm{Exp}(L). \tag{11}$$

In order to create the canonical SNF, the R-operation removes different number of pairs of the identical cubes

$$\mathrm{SNF(M)} = \mathrm{R(Exp(M))} = \mathrm{R(Exp(L))} = \mathrm{SNF(L)}. \tag{12}$$

Comparing (11) and (12) the number of removed pairs of cubes must be larger in the case of ESOP L.

Reverse the process. Adding certain pairs of identical cubes to SNF(f) leads to different Exp(f). Only in this case of adding the smallest number of pairs of identical cubes to SNF(f) = SNF(M) = SNF(L), the expansion Exp(M) can be created. The collapsing operation Exp^{-1} (the reverse operation of expansion Exp) of Exp(M) calculates the exact minimum ESOP $M = \mathrm{Exp}^{-1}(\mathrm{Exp}(M))$. Q.E.D.

Lemma 5. Two cubes of a reduced ESOP of $f(x_1, x_2, \dots, x_n)$ have the distance D defined as follows:

$$1 \le D \le n. \tag{13}$$

Proof. A reduced ESOP cannot include identical cubes, thus $D \geq 1$. The support of the function *f* is given by *n*. This number *n* is the highest number of variables, where both cubes may be different. Q.E.D.

Theorem 5. Let two cubes, c_1 and c_2, belonging to ESOP $f(x_1, x_2, \dots, x_n)$, have the distance *D*. Then their expansions, Exp(c_1) and Exp(c_2), overlap in 2^{n-D} cubes.

Proof. According to Lemma 5, the distance *D* cannot be larger than *n*. For each pair of variables in c_1 and c_2, one of the following two cases holds. Either the variables have a distance of zero, which means that there are identical values ((00), (11), (--)), or the variables have a distance of one, that means there appears one of the pairs ((01), (0-), (10), (1-), (-0), (-1)).

Expanding one variable of c_1 leads to the two remaining values in the set {0, 1, -}

$$0 \rightarrow \{1, -\},$$

$$1 \rightarrow \{0, -\},$$

$$- \rightarrow \{0, 1\},$$

Compare these formulas to (5), (6), (7). The same is valid for the corresponding variable of c_2.

Assume the distance $D = 0$. For all *n* variables v_i the first case holds. It is obvious that the expansions Exp(c_1) and Exp(c_2) are the same and overlap in all 2^n cubes. The theorem holds in this case because $2^{n-D} = 2^{n-0} = 2^n$.

Assume further that the distance $D = n$. For all *n* variables v_i, the second case holds. The following list shows that for each type of distance 1 there is exactly one common value in the expansion:

$v_{1i} = 0 \rightarrow \{1, -\}$, $v_{2i} = 1 \rightarrow \{0, -\}$, common value {-}

$v_{1i} = 0 \rightarrow \{1, -\}$, $v_{2i} = - \rightarrow \{0, 1\}$, common value {1}

$v_{1i} = 1 \rightarrow \{0, -\}$, $v_{2i} = 0 \rightarrow \{1, -\}$, common value {-}

$v_{1i} = 1 \rightarrow \{0, -\}$, $v_{2i} = - \rightarrow \{0, 1\}$, common value {0}

$v_{1i} = - \rightarrow \{0, 1\}$, $v_{2i} = 0 \rightarrow \{1, -\}$, common value {1}

$v_{1i} = - \rightarrow \{0, 1\}$, $v_{2i} = 1 \rightarrow \{0, -\}$, common value {0},

thus the expansions Exp(c_1) and Exp(c_2) overlap in only one cube. This cube is given by the above listed common values. The theorem holds in this case, because $2^{n-D} = 2^{n-n} = 2^0 = 1$.

If $0 < D < n$, the cubes c_1 and c_2 include *D* variables v_i having distance 1 and the remaining $n - D$ variables v_j with distance 0, respectively. The following list shows that, for each type of distance 0 between the corresponding variables, there are exactly two common values in the expansion:

$v_{1j} = 0 \rightarrow \{1, -\}$, $v_{2j} = 0 \rightarrow \{1, -\}$, common value {1, -}

$v_{1j} = 1 \rightarrow \{0, -\}$, $v_{2j} = 1 \rightarrow \{0, -\}$, common value {0, -}

$v_{1j} = - \rightarrow \{0, 1\}$, $v_{2j} = - \rightarrow \{0, 1\}$, common value {1, 0}.

Decreasing the distance *D* by 1 doubles the number of common values for one variable and, consequently, doubles the number of cubes where the expansions Exp(c_1) and Exp(c_2) overlap. Because there is *one* common value for *D* variables v_i and $n - D$ variables v_j having *two* common values, the number of cubes, in which expansions Exp(c_1) and Exp(c_2) overlap, is $1^D * 2^{n-D} = 2^{n-D}$. Q.E.D.

Figure 4 illustrates how the fundamental property of lattices is transferred to the adjacency graph of the SNF.

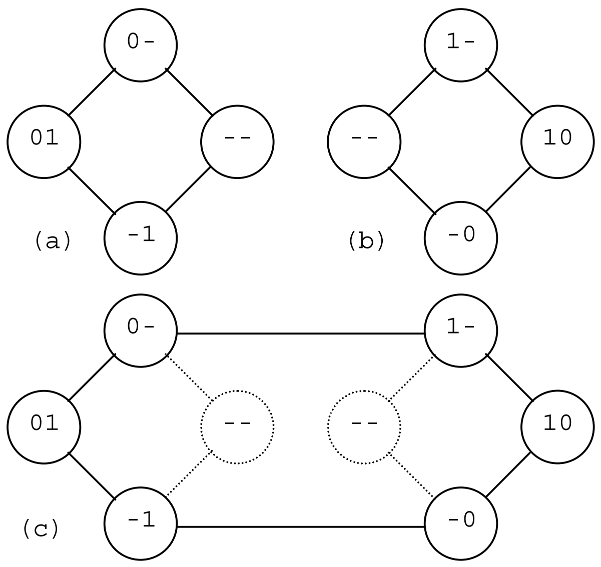


Figure 4. (a) Exp($a\bar{b}$), (b) Exp($\bar{a}b$), (c) SNF($a\bar{b} \oplus \bar{a}b$)

Lemma 6. The expansions L_1 = Exp(c_1) of a cube c_1 having the support of *n* variables is a lattice and the degree of any vertex (the number of adjacency edges of the vertex) in the adjacency graph *AG*(*V*, *E*) of L_1 is equal to *n*.

Proof. The number of cubes in L_1 is 2^n. These cubes are created by all possible combination of the two allowed values of each variable. For each cube in L_1, it is true that changing the value for each of the *n* variables separately leads to another cube of the lattice. The cubes of the lattice are associated with the vertices of the adjacency graph *AG*(*V*, *E*) and each vertex has the degree *n* by construction. Q.E.D.

Lemma 7. Assume that the ESOP of $f(x_1, x_2, \dots, x_n)$ includes only two cubes c_1 and c_2 and the lattices of their expansions L_1 = Exp(c_1) and L_2 = Exp(c_2) overlap exactly in one cube. The degree of any vertex of the adjacency graph *AG*(*V*, *E*) of SFN($c_1 \oplus c_2$) is equal to *n*.

Proof. According to Lemma 6, each vertex of the adjacency graphs $AG(V, E)$ of the expansions $L_1 = \text{Exp}(c_1)$ and $L_2 = \text{Exp}(c_2)$ has the degree n. The R-operation $\text{R}(\text{Exp}(c_1) \oplus \text{Exp}(c_2))$ removes the cube, in which L_1 and L_2 overlap, in order to create $\text{SFN}(c_1 \oplus c_2)$. The removed vertices are connected to n distance-1 vertices in the lattices. According to (1), there exist n pairs of vertices that are distance-1 in exactly one variable. These n pairs of vertices must be connected according to the definition of the adjacency graph. In each of these pairs, one edge is removed and one edge is added, and therefore the vertex degrees do not change. Q.E.D.

Theorem 6. The degree (the number of adjacent edges) of any vertex in the adjacency graph $AG(V, E)$ of the $\text{SNF}(f(x_1, x_2, \dots, x_n))$ is equal to n.

Proof. The $\text{SNF}(f)$ is a canonical ESOP of the function f. Therefore, if c is a cube and $f = f_0 \oplus c$,

$$\text{SNF}(f) = \text{SNF}(f_0 \oplus c) = \text{R}(\text{Exp}(f_0 \oplus c)). \qquad (14)$$

The expansion operation Exp( ) is linear:

$$\text{Exp}(f_a \oplus f_b) = \text{Exp}(f_a) \oplus \text{Exp}(f_b). \qquad (15)$$

Using (15) in (14) leads to

$$\text{SNF}(f) = \text{SNF}(f_0 \oplus c) = \text{R}(\text{Exp}(f_0) \oplus \text{Exp}(c)). \qquad (16)$$

The R-operation is idempotent:

$$\text{R}(f_a) = \text{R}(\text{R}(f_a)). \qquad (17)$$

Using (17) in (16) leads to

$$\text{SNF}(f) = \text{SNF}(f_0 \oplus c) = \text{R}(\text{R}(\text{Exp}(f_0)) \oplus \text{Exp}(c)) \qquad (18)$$

and finaly to (19)

$$\text{SNF}(f) = \text{SNF}(f_0 \oplus c) = \text{R}(\text{SNF}(f_0) \oplus \text{Exp}(c)). \qquad (19)$$

The formula (19) shows that the SNF can be computed recursively. Expanding the first cube leads to a lattice. According to Lemma 6, the degree of vertices of the corresponding adjacency graph $AG(V, E)$ is n. Adding the expansion of the second cube, one or more vertices of $AG(V, E)$ overlap and will be removed by the R-operation. If only one vertex overlaps, the degree of vertices of the corresponding adjacency graph $AG(V, E)$ is n, according to Lemma 7. If there is an overlap in more than one vertex between the $AG(V, E)$ of the previous SNF, the following considerations show that the degree of all vertices of the created adjacency graph $AG(V, E)$ is n.

Take one vertex from the expanted cube that overlaps with the cube from the previous SNF. Both associated cubes have n distance-1 cubes. Consider two distance-1 cubes w.r.t. variable v_i. These two cubes may be identical or have distance 1, too. In the first case, both associated vertices are removed. If the selected three cubes, the overlapping cube and the two distance-1 cubes w.r.t. variable v_i, cover in this variable all possible values {0, 1, -}, the edges to the overlapping cube are removed and one new edge between the two distance-1 is created (see Figure 4). Thus, the degree of all remaining vertices keeps n. Q.E.D.

5 Applications of SNF

5.1 Overview of the Basic Strategy

There are several applications of the SNF. These applications are closely related to the EXOR operation, because the SNF is a reduced ESOP.

The number of cubes in the SNF is typically high because the expansion Exp-operation creates 2^n cubes form each cube in the given ESOP. The R-operation reduces this number, but, according to Theorem 5, in the worst case, only one pair of cubes, out of the $2 * 2^n$ cubes in the expansion of two given cubes, can be removed. However, in general the SNF is not the largest reduced ESOP of the represented function f.

Now we describe the basic strategy how the SNF can be used to find the minimum ESOP(s). This strategy is schematically represented in Figure 5.

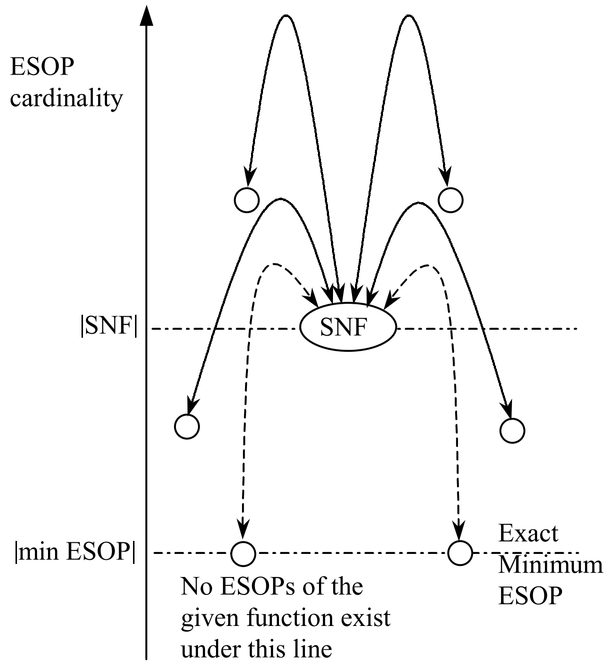


Figure 5. Schematic representation of SNF-based ESOP minimization strategy.

The bubbles in Figure 5 show individual ESOPs of the function. The vertical axis shows the direction of the increase in the number of cubes in an ESOP. Curvy lines show trajectories in the imaginary space of all ESOP traced by the functional representations during minimization.

Minimization is performed in two steps. First, the SNF is constructed starting from any known ESOP (or computed from another functional representations such as BDDs). Because the SNF is canonical, during this first step the knowledge about the given ESOP is lost. It is known, that the R-operation removes the smallest number of cubes if the Exp-operation expands an exact minimum ESOP. Therefore, in the second step a way from SNF to the exact minimum ESOP is found by adding the smallest number of cubes to the SNF (Theorem 4). The cubes of the exact minimum ESOP are found by collapsing complete lattices of cubes, each composed of 2^n cubes. Schematically, the exact minimum ESOP is positioned on the lower bound of the functional space containing all the ESOPs of the given function (Figure 5).

The advantage of the proposed algorithm is that no top down search for the smallest ESOP is necessary. Only a small number of pairs of cubes must be added to fill up the embedded parts of lattices of the SNF to complete lattices, instead. The number of pairs n_{pc} of cubes which must be added is at least

$$n_{pc} = ((\, 2^n - (|\, \mathrm{SNF}(\, M\,)\,|\bmod 2^n\,))\bmod 2^n\,)\,/\,2 \qquad (20)$$

because if the expansion containing $k * 2^n$ ($k \geq 0$) cubes exists, this expansion with the smallest k can be used to get the exact minimum ESOP by collapsing the SNF cubes. If it is not possible to create complete lattices from the SFN, n_{pc} must enlarged by 2^{n-1} cubes one or more times.

5.2 An Approach to Exact ESOP Minization Using Cube Grouping

Not only the number of pairs n_{pc} of cubes, which must be added, is known, the pairs themselves can be found using the SNF, too. The following example shows how this knowledge can be found in the SNF.

Example 1

Assume a Boolean function f depends on the four variables a, b, c, and d. Their SNF includes 40 cubes listed below.

SNF(f) =

```
abcd   abcd   abcd
----   0-00   1---
---1   0-01   1--1
--00   0-10   1-1-
--01   0-11   1-11
--1-   00--   10--
--10   00-0   10-0
-000   0000   101-
-001   0001   1010
-010   001-   11-0
-011   0011   11-1
-1--   01--   1110
-1-1   01-0   1111
-11-   011-
-111   0110
```

The set of cubes of a lattice may be created recursively. Taking two different values for one variable forms the simplest lattice. Taking again two different values for the next variable, connect each of them to the elements of the previous lattice and put all cubes together forms the lattice having one more variable. The reverse procedure helps to find the missing pairs of cubes in the SNF needed to create the separate complete lattices. In the SNF, such pairs of cubes are selected, which have different values for the first variable and the identical values for the remaining variables. For the given SNF(f), we get the following 20 pair of cubes divided in 3 groups, characterized by the combination of the values in the first variable.

```
Cube group a = [01]:
          --11
          -0--
          -0-0
          -01-
          -1-0
          -110

Cube group a = [0-]:
          --00
          --01
          --10
          -000
          -001
          -011
          -1--
          -11-

Cube group a = [1-]:
          ----
          ---1
          --1-
          -010
          -1-1
          -111
```

There is no cube in the SNF which can not combined with an other cube of the SNF to such a pair. The above procedure may be repeated for the second variable b and leads to following tree of cubes.

```
Cube group a = [01]:
     Cube group b = [01]:
               ---0
     Cube group b = [0-]:
     Cube group b = [1-]:
     Cube group 4 (remainder):
               --11
               -0--
               -01-
               -110
```

```
Cube group a = [0-]:
     Cube group b = [01]:
     Cube group b = [0-]:

               --00
               --01
     Cube group b = [1-]:
     Cube group 4 (remainder):

               --10
               -011
               -1--
               -11-

Cube group a = [1-]:
     Cube group b = [01]:
     Cube group b = [0-]:
     Cube group b = [1-]:

               ---1
     Cube group 4 (remainder):

               ----
               --1-
               -010
               -111
```

This distribution shows what cube pairs must be added to create a complete lattice. Note that there are some empty groups. No cube pairs of the associated lattice are includes in the SNF. In the groups labeled by 4, the cubes are collected, which do not form a pair. In order to create complete lattices, a number of cube pairs must be added, which move the cubes from the groups labeled by 4. The first cube in *a* = `[01]` group 4 `(--11)` represents the pair of cubes `((0-11)(1-11))`. In order to move this cube into two other groups, a pair of identical cubes created by the third possible value in the first column `a = [-]` and the values of the selected cube in the remaining columns are necessary. So, the first pair of cubes to add is **`((--11)(--11))`**. Now these four cubes can be merged into two new pairs of cubes `((0-11)(--11))` and `((1-11)(--11))`. The first pair has a `[0-]` combination for the variable *a* and therefore moves to the second group. Inside of this group there already exists the pair of cubes `((0011)(-011))`. Therefore, using the first cube of the added pair of cubes, the group *a* = `[0-]`, *b* = `[0-]` is extended by the cube `(--11)`. The second merged pair `((1-11)(--11))` has `[1-]` combination for the variable *a* and therefore moves to the third group. Inside this group, there already exists the pair of cubes `((1111)(-111))`. Therefore, using the second cube of the added pair of cubes, the group *a* = `[1-]` and *b* = `[1-]` is extended by the cube `(--11)`. Note that adding one pair of identical cubes reduces the number of cubes in each of the three cube groups 4 by one. These three cubes have identical values for the variables *c* and *d*. For the variable *b*, all values of the set {`0, 1, -`} are present. In fact, there are three choices for the selection of the cube pairs that should be added. Only the selected pair of cubes moves the mentioned three cubes from the groups 4 into a non-empty group. After adding the pair of cubes `((--11)(--11))`, the following group structure is created.

```
Cube group a = [01]:
     Cube group b = [01]:
               ---0
     Cube group b = [0-]:
     Cube group b = [1-]:
     Cube group 4 (remainder):
               -0--
               -01-
               -110
Cube group a = [0-]:
     Cube group b = [01]:
     Cube group b = [0-]:
               --00
               --01
               --11
    Cube group b = [1-]:
    Cube group 4 (remainder):
               --10
               -1--
               -11-
Cube group a = [1-]:
     Cube group b = [01]:
     Cube group b = [0-]:
     Cube group b = [1-]:
               ---1
               --11
     Cube group 4 (remainder):
               ----
               --1-
               -010
```

Applying a similar procedure to the cube pair
`((01--)(-1--))`
the second pair of identical cubes

`((11--)(11--))`

is added, which extends each of the groups *a* = `[01]`, *b* = `[01]` and *a* = `[1-]`, *b* = `[1-]` by the cube `(----)`.

Similarly to the pair of cubes
`((011-)(-11-))`
in second group the pair of identical cubes

`((111-)(111-))`

is added, which extends each of the groups *a* = `[01]`, *b* = `[01]` and *a* = `[1-]`, *b* = `[1-]` by the cube `(--1-)`.

Finally, to the pair of cubes
`((1010)(-010))`
in third group the pair of identical cubes

`((0010)(0010))`

is added, which extends each of the groups a = `[01]`, b = `[01]` and a = `[0-]`, b = `[0-]` by the cube `(--10)`. After adding these four pair of cubes the following group structure is created.

```
Cube group a = [01]:
     Cube group b = [01]:

               ---0
               ----
               --1-
               --10
     Cube group b = [0-]:
     Cube group b = [1-]:
     Cube group 4 (remainder):

Cube group a = [0-]:
     Cube group b = [01]:
     Cube group b = [0-]:

               --00
               --01
               --11
               --10
    Cube group b = [1-]:
    Cube group 4 (remainder):

Cube group a = [1-]:
     Cube group b = [01]:
     Cube group b = [0-]:
     Cube group b = [1-]:

               ---1
               --11
               ----
               --1-
     Cube group 4 (remainder):
```

Now the procedure of dividing into groups is repeated for variables c and d. Skipping empty groups we get the following group structure.

```
Cube group a = [01]:
     Cube group b = [01]:
          Cube group c = [1-]:
               Cube group d = [0-]:
               ----

Cube group a = [0-]:
     Cube group b = [0-]:
          Cube group c = [01]:
               Cube group d = [01]:
               ----

Cube group a = [1-]:
     Cube group b = [1-]:
          Cube group c = [1-]:
               Cube group d = [1-]:
               ----
```

By this group structure three lattices are found, which can be collapsed into three cubes of the minimum ESOP.

	abcd		abcd		abcd
	0010		0000		1111
	001-		0001		**111-**
	00-0		**0010**		11-1
	00--		0011		**11--**
	0110		0-00		1-11
	011-		0-01		1-1-
	01-0		0-10		1--1
L1 =	01--	L2 =	0-11	L3 =	1---
	1010		-000		-111
	101-		-001		-11-
	10-0		-010		-1-1
	10--		-011		-1--
	1110		--00		**--11**
	111-		--01		--1-
	11-0		--10		---1
	11--		**--11**		----
c1 =	--01	c2 =	11--	c3 =	0000

Observe that the above three lattices include all 40 cubes of the SNF and four pairs of identical cubes printed in bold letters. The found ESOP is:

$$f = \bar{c}d \oplus ab \oplus \bar{a}\,\bar{b}\,\bar{c}\,\bar{d}\,. \qquad (21)$$

(end of Example 1)

5.3 Proof of Exactness Using a Loose Lower Bound

If a small ESOP like (21) for a Boolean function is found, the question arises: Is this ESOP the smallest one? The SNF helps to answer to this question. According to Theorem 4, the function f of (21) is an exact minimum ESOP if the smallest number of pairs of identical cubes was added to the SNF(f). In Example 1, four such pairs were added. The smallest number of pair that must be added to the SNF is defined by (20). The number of cubes in the SNF of Example 1 is | SNF(f) | = 40 and the number of variables is $n = 4$.

$$n_{pc} = ((\,2^n - (|\,\mathrm{SNF}(\,M\,)\,|\bmod 2^n\,))\bmod 2^n\,)\,/\,2$$
$$n_{pc} = ((16 - (40 \bmod 16)) \bmod 16) / 2$$
$$n_{pc} = ((16 - (8)) \bmod 16) / 2$$
$$n_{pc} = ((8) \bmod 16) / 2$$
$$n_{pc} = (8) / 2$$
$$n_{pc} = 4$$

Both the number of necessary pairs n_{pc} to add to SNF(f) and the real number of pair of cubes added in Example 1 are 4, therefore the ESOP (21) is an exact minimum of the function f. This answer may be also found using the weak lower bound defined in Theorem 4, which tells that there can not exist an ESOP of an function f having less than k

cubes. For the function f of the Example 1, characterized by $|\,\mathrm{SNF}(f)| = 40$ and $n = 4$, the lower bound is

$k = \lceil\, |\,\mathrm{SNF}(f)\,| \,/\, 2^n \,\rceil$

$k = \lceil 40 \,/\, 16 \,\rceil$

$k = \lceil 2.5 \,\rceil$

$k = 3.$

ESOP (21) includes 3 cubes, thus, this ESOP is an exact minimum.

5.4 An Approach to Exact ESOP Minization Using Coverage Matrix

It is known, that on average the number of cubes in an ESOP is smaller than in a SOP, but for certain functions the minimum SOP contains fewer cubes than the minimum ESOP. According to Theorem 2, the number of different reduced ESOPs of each Boolean function is $2^{(3^n)-(2^n)}$. This set of functions includes all exact minimum ESOPs, some nearly minimum ESOPs and further larger ESOPs. If several Boolean functions should be realized, the global minimum strongly depends on the number of reused cubes. In this context a quantitative evaluation is important, which tells us what is the likelihood that the cube belongs to the minimum ESOP. The answer to this question can be found using the SNF.

In order to evaluate each cube of a given support w.r.t. to the SNF(f), a matrix similar to Table 2 can be created. The rows are labeled by all possible 3^n cubes and the columns are labeled by the cubes of the SNF(f). The basic algorithm to find an ESOP containing a small number of cubes is to add some pairs of identical cubes, which do not change the function. As a result, separate complete lattices will be created. Collapsing these lattices gives the minimum ESOP. The cubes of these lattice are therefore the cubes from the SNF and the added pairs of cubes.

The question is, how many and which lattices can cover a certain cube form the SNF? From the algorithm to perform operation Exp(c_0), the value of a selected variable in the cube c_s of the SFN is different from the value of the previous cube c_0. Vice versa, the value of a selected variable in the cube c_s of the SFN can only be created by the other two values of the set `{0, 1, -}`. Thus, exactly 2^n of the 3^n cubes can cover a selected cube from the SNF. That means at least one of these 2^n cubes must be a cube of the solution. The value 1 in Table 2 means that the cube of the row covers one cube in the allowed lattice of a SNF-cube of the column.

Observe, for instance, that the SNF-cube (`----`) in the first column of Table 2 can be covered by that 16 cubes, which are build by the values zero and one only. Note, the number of ones in each column of Table 2 is constant and equal to 16, because the support number of variables in this

Table 2. Coverage matrix of a SNF

```
    a ----------------0000000000000000111111111111
    b ------0000111111------0000001111000000111111
    c --0011--00--0011--0011--0011--00--0011--0011
    d -1-0010101-0-101-1-001-0-1010101-1-001-1-001
abcd                                                w
----  00000000000000000000000001110011000111000111 11
---0  00000000000000000000000011010001001001001001  8
---1  00000000000000000000000010100010001110001110  9
--0-  00000000000000000000000100111100010011010011 11
--00  00000000000000000000001000010100110001110001  9
--01  00000000000000000000001100101000100010100010  8
--1-  00000000000000000000000101001111010100010100 10
--10  00000000000000000000001011000101111000111000 11
--11  00000000000000000000001110001010101100101100 11
-0--  00000000000000000001110000000011000000000111  8
-0-0  00000000000000000010010000000001000000001001  5
-0-1  00000000000000000011100000000010000000001110  7
-00-  00000000000000000100110000001100000000010011  8
-000  00000000000000001100010000000100000000110001  7
-001  00000000000000001000100000001000000000100010  5
-01-  00000000000000000101000000001111000000010100  8
-010  00000000000000001110000000000101000000111000  8
-011  00000000000000001011000000001010000000101100  8
-1--  00000000000000000001110001110000000111000000  9
-1-0  00000000000000000010010011010000001001000000  7
-1-1  00000000000000000011100010100000001110000000  8
-10-  00000000000000000100110100110000010011000000  9
-100  00000000000000001100011000010000110001000000  8
-101  00000000000000001000101100100000100010000000  7
-11-  00000000000000000101000101000000010100000000  6
-110  00000000000000001110001011000000111000000000  9
-111  00000000000000001011001110000000101100000000  9
0---  00000000110001110000000000000000000111000111 11
0--0  00000000010011010000000000000000001001001001  8
0--1  00000000100010100000000000000000001110001110  9
0-0-  00000011000100110000000000000000010011010011 11
0-00  00000001001000010000000000000000110001110001  9
0-01  00000010001100100000000000000000100010100010  8
0-1-  00000011110101000000000000000000010100010100 10
0-10  00000001011011000000000000000000111000111000 11
0-11  00000010101110000000000000000000101100101100 11
00--  00011100000001110000000000000000000000000111  9
00-0  00100100000011010000000000000000000000001001  7
00-1  00111000000010100000000000000000000000001110  8
000-  01001100000100110000000000000000000000010011  9
0000  11000100001000010000000000000000000000110001  8
0001  10001000001100100000000000000000000000100010  7
001-  01010000000101000000000000000000000000010100  6
0010  11100000001011000000000000000000000000111000  9
0011  10110000001110000000000000000000000000101100  9
01--  00011100110000000000000000000000000111000000  8
01-0  00100100010000000000000000000000001001000000  5
01-1  00111000100000000000000000000000001110000000  7
010-  01001111000000000000000000000000010011000000  8
0100  11000101000000000000000000000000110001000000  7
0101  10001010000000000000000000000000100010000000  5
011-  01010011110000000000000000000000010100000000  8
0110  11100001010000000000000000000000111000000000  8
0111  10110010100000000000000000000000101100000000  8
1---  00000000110001110000000001110011000000000000 10
1--0  00000000010011010000000011010001000000000000  8
1--1  00000000100010100000000010100010000000000000  6
1-0-  00000011000100110000000100111100000000000000 10
1-00  00000001001000010000001000010100000000000000  6
1-01  00000010001100100000001100101000000000000000  8
1-1-  00000011110101000000000101001111000000000000 12
1-10  00000001011011000000001011000101000000000000 10
1-11  00000010101110000000001110001010000000000000 10
10--  00011100000001110001110000000011000000000000 11
10-0  00100100000011010010010000000001000000000000  8
10-1  00111000000010100011100000000010000000000000  9
100-  01001100000100110100110000001100000000000000 11
1000  11000100001000011100010000000100000000000000  9
1001  10001000001100101000100000001000000000000000  8
101-  01010000000101000101000000001111000000000000 10
1010  11100000001011001110000000000101000000000000 11
1011  10110000001110001011000000001010000000000000 11
11--  00011100110000000001110001110000000000000000 11
11-0  00100100010000000010010011010000000000000000  8
11-1  00111000100000000011100010100000000000000000  9
110-  01001111000000000100110100110000000000000000 11
1100  11000101000000001100011000010000000000000000  9
1101  10001010000000001000101100100000000000000000  8
111-  01010011110000000101000101000000000000000000 10
1110  11100001010000001110001011000000000000000000 11
1111  10110010100000001011001110000000000000000000 11
```

example is equal to four. The numbers of ones in the rows are not constant. The sum of the ones in the rows is expressed in the last column of Table 2 and labeled by *w*, that means the *weight* of the cube in the row. The weight *w* tells us, how many cubes from the SNF are covered by the lattice created from the cube in the row. The larger the weight *w*, the more likely the cube belong to the minimum ESOP. These weights *w* can used to the select the cubes to find a global minimum of a set of functions. Based on the weights *w*, a simple minimization algorithm can defined.

(1) Take the cube (one of the cubes) having the largest weight *w*.
(2) Add this cube to the solution ESOP.
(3) Expand this cube to a lattice.
(4) Remove the cubes, covered by this lattice from the SNF.
(5) Add the cubes of the lattice, which not match with the SNF.
(6) If the SNF is not empty, continue with (1).

This simple greedy algorithm has the disadvantage, that in each sweep only one locally optimal cube is selected. In the example of Table 2, there exists one cube `(1-1-)` = *ac,* having the highest weight $w = 12$. Using this cube an ESOP of five cubes can found: $f = ac \oplus \bar{c} \oplus a\bar{b}\bar{c} \oplus cd \oplus a\bar{b}cd$. This is not a minimum ESOP, but the cube *ac* is part of an expression of the function *f* using an EXOR operation and having the smallest literal count: $f = ac \oplus \left(\overline{a\bar{b}} \cdot \overline{c\bar{d}}\right)$.

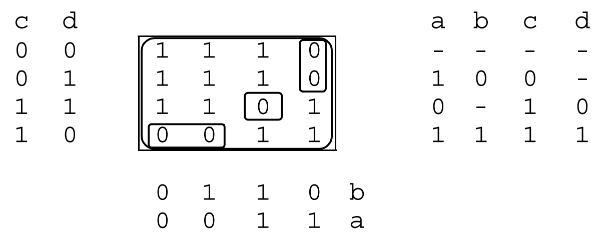
5.5 Proof of Exactness Using a Tight Lower Bound

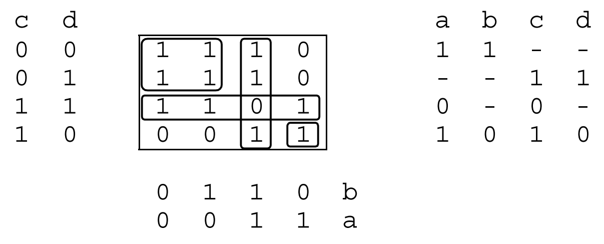
The Table 2 includes 16 cubes of the second highest weight $w = 11$. All of them can be taken to create an ESOP of four cubes, shown in Figure 6. Are there four ESOPs exact minimum ESOPs? From Table 2, we get the number of cubes in this SNF to be $|\mathrm{SNF}(f)| = 44$ and the size of the support $n = 4$. According to Theorem 3, the weak lower bound can be calculated as follows:

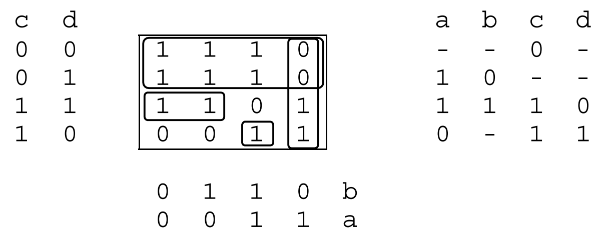
$$k = \lceil\, |\mathrm{SNF}(f)| / 2^n \,\rceil$$
$$k = \lceil\, 44 / 2^4 \,\rceil$$
$$k = \lceil\, 44 / 16 \,\rceil$$
$$k = \lceil\, 2.75 \,\rceil$$
$$k = 3.$$

There are four cubes in the ESOP that was found and the weak lower bound is smaller, namely three. Here the question arises, are there any ESOPs of three cubes or can we find a tighter lower bound, to prove that the ESOPs in Figure 6 is the exact minimum. Again, the second question can be answered using the SNF.

The proof of Theorem 3 is based on the size of the lattice 2^n. This is the largest value that can be covered by one cube (one row in Table 2) in the SNF. The weight *w* of cubes covered in the SNF is typically smaller. All cubes of


(1) $f = 1 \oplus a\bar{b}\bar{c} \oplus \bar{a}c\bar{d} \oplus abcd$


(2) $f = ab \oplus cd \oplus \bar{a}\bar{c} \oplus a\bar{b}c\bar{d}$


(3) $f = \bar{c} \oplus a\bar{b} \oplus abc\bar{d} \oplus \bar{a}cd$

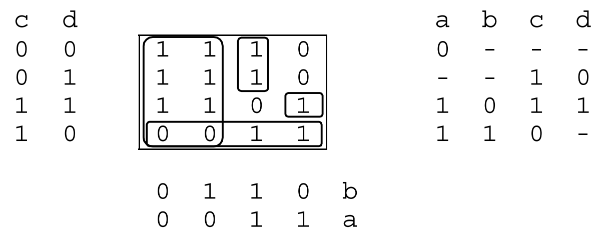

(4) $f = \bar{a} \oplus c\bar{d} \oplus a\bar{b}cd \oplus ab\bar{c}$

Figure 6. Four exact minimum ESOPs of the function form Table 2

the SNF must be covered by the selected cubes of the minimum ESOP. It is not possible to find a minimum ESOP with less cubes than found by the following algorithm.

Algorithm SLB (stronger lower bound)

```
(1)n_slb = 0; m = |SNF( f )|;
(2)if (m <= 0) n_slb is the stronger lower bound
     end
(3)n_slb = n_slb + 1; m = m – w_max
     exclude cube of the selected w_max
     goto (2)
```

In the above algorithm `w_max` is the maximal weight *w* of a cube for the SNF. Appling the algorithm to calculate a stronger lower bound we get in case of the SNF of Table 2 step by step the values of Table 3 and the stronger lower bound of four. Therefore, the ESOPs listed in Figure 6 are exact minimum ESOPs and no ESOP with less than four cubes exist for the SNF(*f*) given in Table 2.

Table 3. Calculation of a stronger lower bound

n_slb	m	w_max
0	44	
1	44 - 12 = 32	12
2	32 - 11 = 21	11
3	21 - 11 = 10	11
4	10 - 11 = -1	11

5.6 Selection of Preferred Cubes

The final application in this paper is focusing on the selection of cubes to cover the cubes of the SNF. From the above approach, it is known that exactly 2^n out of 3^n cubes cover a selected cube from the SNF. The introduced weights *w* helps evaluate these 2^n cubes. Are there a stronger quality criteria to distinguish between more or less useful cubes? Yes, according to Theorem 6, there is one more criterion to get useful knowledge from the SNF.

In order to figure out this property, the SNF of a single cube is considered. The associated SNF is a lattice, created by the expansion operation Exp( ). Each cube of this lattice has n edges to distance-1 cubes. Taking the values of the *n* variables of one cube of the lattice and the changed values in the *n* distance-1 cubes, for each variable, a pair of values is found. The single cube of the exact minimum ESOP of this SNF can reconstructed, taking the missing third value for each variable. Note that this reconstruction is done starting with any cube of the lattice. Example 2 illustrates this approach using the cube $a\bar{b}$ with the lattice in Fig. 4a.

Example 2

Cubes of the lattice of $a\bar{b}$ are:
`((01), (0-), (-1), (--)).`

The reconstruction starting from each cube of a lattice finishes always with the cube $a\bar{b}$

cube of the lattice	distance-1 cubes	combinations of values	reconstructed basic cube
(01)	((-1),(0-))	{0-},{1-}	(10)
(0-)	((--),(01))	{0-},{1-}	(10)
(-1)	((01),(--))	{0-},{1-}	(10)
(--)	((0-),(-1))	{0-},{1-}	(10)

(end of Example 2)

The Example 2 shows how the distributed knowledge about the cube can be reconstructed. For this reconstruction, only a part of the lattice is necessary. In fact, this part is a corner of the lattice, defined by any cube of the lattice and the associated distance-1 cubes.

The first step to construct the SNF of a function *f* is the expansion of the given cubes into lattices. The R-operation removes some cubes from the lattices, so that the distributed knowledge about the given cubes is reduced. Corresponding to the R-operation, in the adjacency graph of the SNF some edges of the intermediate lattices are removed, too, but additional edges are introduced. In a way, by means of the R-operation new corners of the potential lattices are created and new suitable cubes for the minimal ESOPs are build. Consider Figure 4c. Each cube associated with a vertex of the adjacency graph of the SNF can be taken to create a lattice which defines after collapsing a cube of the minimum ESOP. Figure 7 illustrates this approach. It is obvious that the ESOPs found are the exact minimum ESOPs.

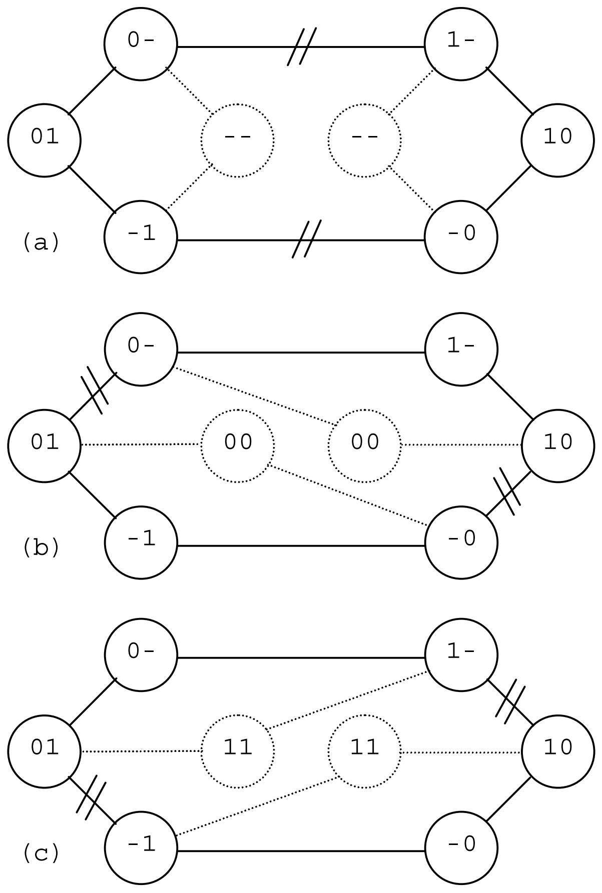


Figure 7. Reconstruction of lattices and ESOPs.
(a) SNF($a\bar{b} \oplus \bar{a}b$), (b) SNF($a \oplus b$), (c) SNF($\bar{a} \oplus \bar{b}$)

Table 4. Preferred cubes defined by a SNF

	a----------------0000000000000000111111111111	
	b------0000111111------0000001111000000111111	
	c--0011--00--0011--0011--0011--00--0011--0011	
	d-1-0010101-0-101-1-001-0-1010101-1-001-1-001	
abcd		w2
----	00000000000000000000000001110011000121000111	1
---0	00000000000000000000000011010001001001001001	0
---1	00000000000000000000000010100010001110001210	1
--0-	00000000000000000000000100111100010012010011	1
--00	00000000000000000000001000010100110001120001	1
--01	00000000000000000000001100101000100010100010	0
--1-	00000000000000000000000101001111010100010100	0
--10	00000000000000000000001011000101211000111000	1
--11	00000000000000000000001110001010102100101100	1
-0--	00000000000000000001110000001100000000000111	0
-0-0	00000000000000000010010000000001000000001001	0
-0-1	00000000000000000011100000000010000000001110	0
-00-	00000000000000000010011000001100000000010011	0
-000	00000000000000001100010000000100000000110001	0
-001	00000000000000001000100000001000000000100010	0
-01-	00000000000000000010100000001221000000010100	2
-010	00000000000000001110000000000101000000111000	0
-011	00000000000000001011000000000101000000101100	0
-1--	00000000000000000001110001120000001110000000	1
-1-0	00000000000000000010010011010000001001000000	0
-1-1	00000000000000000001110001010000001110000000	0
-10-	00000000000000000100110100210000010011000000	1
-100	00000000000000001100011000010000110001000000	0
-101	00000000000000001000101100100000100010000000	0
-11-	00000000000000000101000101000000010100000000	0
-110	00000000000000001110001021000000111000000000	1
-111	00000000000000001011002110000000101100000000	1
0---	00000000110001110000000000000000000111000121	1
0--0	00000000010011010000000000000000001001001001	0
0--1	00000000100010100000000000000000001210001110	1
0-0-	00000011000100110000000000000000010011010012	1
0-00	00000001001000010000000000000000120001110001	1
0-01	00000010001100100000000000000000100010100010	0
0-1-	00000011110101000000000000000000010100010100	0
0-10	00000001011011000000000000000000111000211000	1
0-11	00000010101110000000000000000000101100102100	1
00--	00011100000001120000000000000000000000000111	1
00-0	00100100000011010000000000000000000000001001	0
00-1	00111000000010100000000000000000000000001110	0
000-	01001100000100210000000000000000000000010011	1
0000	11000100001000010000000000000000000000110001	0
0001	10001000001100100000000000000000000000100010	0
001-	01010000000101000000000000000000000000010100	0
0010	11100000001021000000000000000000000000111000	1
0011	10110000002110000000000000000000000000101100	1
01--	00011100110000000000000000000000001110000000	0
01-0	00100100010000000000000000000000001001000000	0
01-1	00111000100000000000000000000000001110000000	0
010-	01001111000000000000000000000000010011000000	0
0100	11000101000000000000000000000000110001000000	0
0101	10001010000000000000000000000000010001000000	0
011-	01010012210000000000000000000000001010000000	2
0110	11100001010000000000000000000000011100000000	0
0111	10110010100000000000000000000000010110000000	0
1---	00000000110001110000000001110011000000000000	0
1--0	00000000010012010000000012010001000000000000	2
1--1	00000000100010100000000010100010000000000000	0
1-0-	00000011000100110000000100111100000000000000	0
1-00	00000001001000010000001000010100000000000000	0
1-01	00000010001200100000000120010100000000000000	2
1-1-	00000211201010000000000101002112000000000000	4
1-10	00000001011011000000001011000101000000000000	0
1-11	00000010101110000000000111000101000000000000	0
10--	00012100000001110001110000000011000000000000	1
10-0	00100100000011010010010000000001000000000000	0
10-1	00111000000010100012100000000001000000000000	1
100-	01001200000100110100110000000110000000000000	1
1000	11000100001000011200010000000100000000000000	1
1001	10001000001100101000100000001000000000000000	0
101-	01010000000101000101000000001111000000000000	0
1010	21100000001011001110000000000101000000000000	1
1011	10210000001110001011000000001010000000000000	1
11--	00011100110000000001210001110000000000000000	1
11-0	00100100010000000010010011010000000000000000	0
11-1	00121000100000000011100010100000000000000000	1
110-	01001111000000000100120100110000000000000000	1
1100	12000101000000000110001100001000000000000000	1
1101	10001010000000001000101100100000000000000000	0
111-	01010011100000001010001010000000000000000000	0
1110	11100001010000002110001011000000000000000000	1
1111	10110010100000001021001110000000000000000000	1

According Theorem 6, the number of distance-1 cubes is equal to n for each cube of the SNF of function f having the support of n variables. Thus, each cube of the SNF and the adjacent distance-1 cubes specify exactly the cube preferred in a minimal ESOP. Table 4 shows this property for the same SNF(f) used in Table 2. Value 2 in the matrix of Table 4 means that the cube of the row is the preferred cube of a SNF-cube of the column. Note that each column of Table 4 contains exactly one value 2. Each value 2 in the matrix of Table 4 replaces one value 1 of the matrix of Table 2, because the criterion to define a value 2 is stronger includes the criterion to define the value 1.

Using the preferred cubes, the second weight w_2 can be defined. The weight w_2 of a cube c expresses, how many cubes of the SNF(f) have the cube c as the preferred cube. In the matrix of Table 4, the weight w_2 of a cube associated to a row is the sum of values two in the row. Observe that the cube ac, `(1-1-)` in Table 2 and 4 has the highest weight $w = 12$ and the highest weight $w_2 = 4$ as well. Comparing Figure 6 and Table 4 shows that no cube of the exact minimum ESOPs of this function has weight $w_2 = 0$.

In general, the larger values of the weight w_2 can be observed for larger values of the weight of w; but the weights w and w_2 are not strongly correlated. For instance the cube `(--00)` has the weights $w = 9$ and $w_2 = 1$ and the cube `(--1-)` has the weights $w = 10$ and $w_2 = 0$.

6 SNF Computation: An Efficient Method

The naive approach to compute the SNF follows the definition. It takes any ESOP of the given function and proceeds by expanding each cube using the Exp-operation and creating the union of these expansions while eliminating the duplicated cubes by the R-operation. The number of cubes in SNF is exponential because each expanded cube contains 2^n cubes, where n is the number of input variables. Managing such a large number of cubes may be problematic if cubes are processed explicitly, one cube at a time.

Another approach to compute the SNF is given below. In this approach, the SNF is represented by Zero-suppressed binary Decision Diagrams (ZDDs) and computed recursively from the BDD of the given function without creating an intermediate ESOP and expanding its cubes. The reader is referred to [5][6], where the foundations of ZDD-based cube manipulation are presented.

Let us denote the positive, negative, and don't-care literals of Boolean variable x by $x^{\{0\}}$, $x^{\{1\}}$, and $x^{\{-\}}$. In the formulas below "$\times$" denotes the operation of multiplying out two sets of cubes and symbols "$\cup$","$\cap$", and "–" denote the set-theoretic union, intersection, and difference applied to the cube sets.

The recursive computation of the SNF is based on the following properties:

Property 1. Let functions f and g depend on the same input variables. Let f_{SNF} and g_{SNF} be the SNFs of f and g. The SNF of $f \oplus g$ is computed by taking the union of the SNFs for functions f and g and removing the duplicated cubes:

$$\text{SNF}(f \oplus g) = (f_{\text{SNF}} \cup g_{\text{SNF}}) - (f_{\text{SNF}} \cap g_{\text{SNF}}). \tag{21}$$

Property 2. Let F_{SNF} be the SNF of function f and x be a Boolean variable that is not currently in the support of f. The SNFs of functions $x^{\{1\}}$ & f, $x^{\{0\}}$ & f, and $x^{\{-\}}$ & f are:

$$\text{SNF}(\, x^{\{1\}} \,\&\, f\,) = (\, x^{\{0\}} \times f_{\text{SNF}}\,) \cup (\, x^{\{-\}} \times f_{\text{SNF}}\,), \tag{22}$$

$$\text{SNF}(\, x^{\{0\}} \,\&\, f\,) = (\, x^{\{1\}} \times f_{\text{SNF}}\,) \cup (\, x^{\{-\}} \times f_{\text{SNF}}\,), \tag{23}$$

$$\text{SNF}(\, x^{\{-\}} \,\&\, f\,) = (\, x^{\{0\}} \times f_{\text{SNF}}\,) \cup (\, x^{\{1-\}} \times f_{\text{SNF}}\,), \tag{24}$$

Theorem. Let SNFs of the negative and positive cofactors be $f^0{}_{\text{SNF}}$ and $f^1{}_{\text{SNF}}$. Then the SNF of function f is:

$$\begin{aligned} \text{SNF}(f\,) \;=\; & \text{SNF}(\, x^{\{0\}} \,\&\, [\, f^0{}_{\text{SNF}} - f^2{}_{\text{SNF}}\,]\,) \\ & \cup\ \text{SNF}(\, x^{\{1\}} \,\&\, [\, f^1{}_{\text{SNF}} - f^2{}_{\text{SNF}}\,]\,) \\ & \cup\ \text{SNF}(\, x^{\{-\}} \,\&\, f^2{}_{\text{SNF}}\,), \end{aligned} \tag{25}$$

where

$$f^2{}_{\text{SNF}} = f^0{}_{\text{SNF}} \cap f^1{}_{\text{SNF}}. \tag{26}$$

Proof. If the SNFs of the two cofactors are known, it is possible to find the common cubes in these SNFs ($f^2{}_{\text{SNF}}$). These cubes correspond to the intersection of cofactors in the given function and go with the empty literal ($x^{\{-\}}$) in the resulting SNF. These cubes should be subtracted from the cubes in $f^0{}_{\text{SNF}}$ and $f^1{}_{\text{SNF}}$ to get the cubes, which go with the negative and positive literals ($x^{\{0\}}$ and $x^{\{1\}}$). Finally, all the three groups of cubes should be added together to create the SNF of the given function. Q.E.D.

The algorithm shown in Figure 8 implements the above formula in a bottom-up traversal of the BDD representing the function. If the standard decision diagram caching techniques are used to store the intermediate results of computation, the complexity of this algorithm is proportional to the number of nodes in the BDD.

The procedure takes two arguments, the function and a set of variables that should appear in the cubes of SNF. Notice that the function may not explicitly depend on some of them.

The terminal cases are trivial. If the function is a constant zero, the SNF is a empty cube set. If no variables are left in the variable set, the function should be constant 1 and the tautology cube is returned.

The fine point of this algorithm is the need to keep track of variables missing in the support of the original function or intermediate subfunctions. These variables are don't-cares on the BDD paths. Although they do not contribute nodes to the BDD, they add new cubes to SNF (and new nodes to the ZDD representing the SNF) according to formula for SNF($x^{\{-\}}$ & f) in Property 2. This situation is taken care of in the first part of the conditional statement. The pseudo-code after "else" implements formula from the Theorem 7 and property 2.

```
cubeset SNF( func F, varset V )
{
   if ( F = 0 )  return ∅;
   if ( V = ∅ ) {assert(F = 1); return {1}};
   x = Var( V );
   if ( x ∉ support(F) ) {
      C = SNF( F, V-x );
      R = (x^{0}×C) ∪ (x^{1}×C);
   }
   else /* if ( x ∉ support(F) ) */ {
      (F0, F1) = Cofactors( F, x );
      C0 = SNF( F0, V-x );
      C1 = SNF( F1, V-x );
      S2 = C0 ∩ C1;
      S0 = C0 − S2;
      S1 = C1 − S2;
      R0 = (x^{1}×S0) ∪ (x^{-}×S0);
      R1 = (x^{0}×S1) ∪ (x^{-}×S1);
      R2 = (x^{0}×S2) ∪ (x^{1}×S2);
      R  = R0 ∪ R1 ∪ R2;
   }
   return R;
}
```

Figure 8. Algorithm for implicit SNF computation.

8 Experimental Results

Table 5 shows experimental results of computing SNF for single-output functions. "Name" is the name of the benchmark. "Out" is the primary output number, for which SNF is computed. "Ins" is the number of primary inputs of the function (some of them may not belong to the support of the given primary output). Column "|SNF|" gives the number of cubes in SNF. Columns N_{BDD} and N_{SNF} give the number of nodes in the BDD of the given function and in the ZDD of SNF. "Time" shows the time needed to compute SNF on a Pentium 300Mhz PC.

Table 5. Experimental result of computing SNF for selected benchmark functions.

name	Out	ins	|SNF|	N_{BDD}	N_{SNF}	time, s	|ESOP|	|SOP|	|primes|
9sym	1	9	9660	25	146	0.01	51	84	1680
alu4	5	14	396824	151	1921	0.11	52	181	381
T481	1	16	533536	202	744	0.05	13	481	481
cordic	1	23	$1.1e10^9$	41	484	0.01	771	143	203
dalu	1	75	$1.6e10^{24}$	107	4529	0.06	96	180	742

To get the sense of how large in the number of cubes in SNF, Table 5 lists the cardinality of three other important cubes sets. Column "|ESOP|" gives the number of cubes in an ESOP minimized heuristically by Exorcism-4 [7]. Column "|SOP|" gives the number of cubes in the exact minimum SOP, and column "|primes|" gives the number of all prime implicants of the given function.

The second set of experimental results is summarized in Table 6. We generated Boolean functions of a given support size (*n*) in such a way that each pair of cubes is orthogonal. The number of orthogonal cubes, |Orth|, are listed in the second column. The set of orthogonal cubes of the function can be connected by EXOR- or OR-operation without changing of the function. The column |SNF| gives the number of the cubes in the SNF. According to Theorem 3, the value of the weak lower bound has been computed and listed in column WLB. Using the algorithm SLB given in Section 5, a stronger lower bound has been calculated. The column SLB in Table 6 shows that this lower bound is significantly larger. The last column of Table 6 gives the number of cubes calculated by a simple minimization algorithm based only on the weight *w*. The comparison to the column SLB shows that in all cases the exact minimum ESOPs has been found.

Table 6. Experimental result of computing exact minimum ESOPs for random functions.

n	|Orth|	|SNF|	WLB	SLB	|EEMIN|
4	4	38	3	3	3
5	4	86	3	4	4
5	10	126	4	6	6
6	17	316	5	8	8
6	13	320	6	8	8
7	19	890	7	11	11
7	18	718	6	9	9
8	12	1822	8	11	11

8 Conclusions

We introduced a new canonical representation for completely specified Boolean functions called Specialized Normal Form (SNF). The SNF has a number of remarkable properties, in particular, it is useful for creating a new approach to exact ESOP minimization, which is not limited by the number of variables in the function.

We propose an efficient algorithm to compute the SNF and an illustrative algorithm for exact ESOP minimization. Even though the current version of the algorithm has complexity exponential in the number of input variables, it has successfully found exact minima for functions of up to nine variables.

Another possible use of SNF is to prove exactness of ESOPs computed by heuristic algorithms.

Our future work is focusing on extracting more information from the SNF in order to find all exact minimum ESOPs of the given function without any search. Another possible improvement is to develop a exact ESOP minimization algorithm with complexity proportional to the size of the ZDD representation of the SNF rather then the number of cubes in SNF.

9 References

[1] M. Escobar, F. Somenzi. Synthesis of AND-EXOR Expressions via Satisfiability. *Proc. of IWLS'95*, Section 8, pp. 1-10.

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

[3] N. Koda, T. Sasao, LP Characteristic Vector for Logic Functions. *Proc. of IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 1993, Wilhelm Schickard-Institute fuer Informatik, pp. 99-108.

[4] N. Koda, T. Sasao. An upper bound on the number of products in minimum ESOPs. *Proc. of IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 1995, Chiba, Japan, pp. 94-101.

[5] S. Minato. *Binary Decision Diagrams and Applications for VLSI CAD.* Kluwer 1995.

[6] S. Minato. Fast Factorization Method for Implicit Cube Cover Representation. *IEEE Trans. CAD*, Vol. 15, No 4, April 1996, pp. 377-384.

[7] A. Mishchenko, M. Perkowski. Fast Heuristic Minimization of Exclusive-Sums-of-Products. Submitted to *5th International Reed-Muller Workshop,* Starkville, Mississippi, August, 2001.

[8] H. Ochi. An Exact Minimization of AND-EXOR Expressions Using Encoded MRCF. IEICE Trans. Fundamentals, Vol. E79-A, No. 12, December 1996, pp. 2131-2133.

[9] M. A. Perkowski, M. Helliwell, P. Wu. Minimization of Multiple-Output Mixed-Radix Exclusive Sums of Produces for Incompletely Specified Functions. *Proc. ISMVL'89*, pp. 256-263.

[10] 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, Wilhelm Schickard-Institute fuer Informatik, pp. 91-98.

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

[12] B. Steinbach,: XBOOLE - A Toolbox for Modelling, Simulation, and Analysis of Large Digital Systems. System Analysis and Modelling Simulation, *Gordon & Breach Science Publishers*, Vol. 9, Number 4, 1992, pp. 297 – 312.