Large-Scale SOP Minimization Using Decomposition and Functional Properties

Alan Mishchenko
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1770
alanmi@eecs.berkeley.edu

Tsutomu Sasao
Center for Microelectronic Systems and Department of CSE
Kyushu Institute of Technology
Iizuka, Fukuoka, 820-8502 JAPAN
sasao@cse.kyutech.ac.jp

ABSTRACT
In some cases, minimum Sum-Of-Products (SOP) expressions of Boolean functions can be derived by detecting decomposition and observing the functional properties such as unateness, instead of applying the classical minimization algorithms. This paper presents a systematic study of such situations and develops a divide-and-conquer algorithm for SOP minimization, which can dramatically reduce the computational effort, without sacrificing the minimality of the solutions. The algorithm is used as a preprocessor to a general-purpose exact or heuristic minimizer, such as ESPRESSO. The experimental results show significant improvements in runtime. The exact solutions for some large MCNC benchmark functions are reported for the first time.

Categories and Subject Descriptors
B.6.3 [**Logic Design**]: Design Aids – Automatic synthesis.

General Terms
Algorithms, Performance, Experimentation, Theory.

Keywords
SOP minimization, disjoint-support decomposition, BDDs, divide-and-conquer strategy, orthodox functions.

1. INTRODUCTION
Exact and heuristic SOP minimization is one of the most well researched problems in the field of computer-aided design. SOP minimization is used in PLA optimization, multi-level logic synthesis, state encoding, power estimation, test generation, and other areas. Due to the exponential nature of the problem of exact SOP minimization, the state-of-the-art algorithms [3][13][7][8] can typically handle functions with up to a hundred products in the minimum SOP. Meanwhile, most of the practical applications and CAD tools rely on heuristic minimization [3][16][6].

DAC 2003, June 2-6, 2003, Anaheim, California, USA.

The complexity of the heuristic algorithms is roughly quadratic in the number of products. These algorithms are noticeably faster than the exact ones but still they can be slow for functions with many products.

Several approaches have been proposed to speed up heuristic SOP minimization. For example, it was observed that computation of the off-set [17] can be time-consuming even for functions with a small number of products in the minimum SOP, such as Achilles' heel function. It was proposed to compute the reduced off-set [11]. Another speedup widely used in the optimization scripts for the logic synthesis tools [22], is to perform only one loop of heuristic minimization. The penalty for such shortcuts is the lower minimization quality, while the runtime problem still remains. For many benchmarks the optimization scripts do not finish because of the long runtime of the heuristic SOP minimization.

Yet another fast heuristic SOP minimization algorithm uses the BDD representation [14]. This algorithm works remarkably well when the quality of the solutions is not critical, for example [10]. However, it was shown [19] that algorithm [14] can produce irredundant SOPs with many more products than minimum SOPs. Therefore, it is not suitable for many practical problems.

In this paper, we formulate several conditions when SOP minimization for completely specified single-output functions can be performed without the time-consuming general-purpose minimization algorithms. One of such properties is the existence of Disjoint-Support Decomposition (DSD). It has been shown in [21] that some form of DSD is present in 75% of benchmark functions, while 33% of them can be decomposed by DSD into canonical networks of two-input gates. Moreover, detecting DSD can be performed efficiently with the complexity polynomial in the number of nodes in the BDD of the benchmark functions using the algorithm [2] with later improvements [12].

The proposed approach to SOP minimization, called MUSASHI. uses a divide-and-conquer strategy, which partitions the minimization problems into a tree of simpler subproblems, and the general-purpose SOP minimizer is used to minimize SOPs at the nodes of this tree. Finally, the SOP is assembled from the partial SOPs using the distributive law. The algorithm works as a preprocessor to the general-purpose SOP minimizer.

The paper is organized as follows. Section 2 introduces the basics. Section 3 presents the theoretical background (proofs are in [24]). Section 4 describes the minimization algorithm. Section 5 shows the experimental results. Section 6 concludes the paper.

2. PRELIMINARIES

2.1 Boolean Functions

In this paper, unless stated otherwise, *function* refers to a completely specified Boolean function $f(X)$: $B^n \rightarrow B$, $B = \{0,1\}$. The *support* of function f is the set of variables X, which influence the output value of f. The support size is denoted by $|X|$. Functions $f(X)$ and $g(Y)$ have *disjoint supports* if $X \cap Y = \varnothing$.

Expressions $\bar{x}$ and x are the *negative literal* and the *positive literal* of variable x, respectively. "Negative" and "positive" are *polarities* of variable x in the corresponding literals. The AND of literals is a *product*.

A product is an *implicant* of f if it implies f, that is, if f is equal to 1 for any assignment of variables that makes the product equal to 1. A *prime implicant*, or *prime*, of f is an implicant of f, if removing any literal from it produces a product that does not imply f. A *minterm* of f is an implicant of f containing all the variables . The *distance* between two minterms is the number of different literals in these minterms.

The OR of implicants of f is a *sum-of-products* (SOP) expression, or a *cover,* of f. The number of products in an SOP C is denoted by $|C|$. An *irredundant sum-of-products expression* (ISOP) of a function f is the OR of primes of f, such that no prime can be deleted without changing f.

A function f is *positive* (*negative*) in variable x if replacing $\bar{x}$ by x (x by $\bar{x}$) in any minterm of f produces a minterm of f. If f is positive (negative) in x, any ISOP of f does not contain products with literal x in the negative (positive) polarity. If f is positive or negative in x, f is *unate* in x.

2.2 SOP Minimization

The problem of *SOP minimization* for f consists in finding an ISOP with the minimum cardinality, among all the ISOPs of f. Such an expression is called a *minimum SOP* (MSOP). In general, a function has many MSOPs. The number of products in an MSOP of f is denoted by $\tau(f)$.

A minterm of f is a *distinguished minterm* if there is exactly one prime of f covering it. Such prime is an *essential prime*. A subset of minterms of f is an *independent set of minterms* of f, if no prime of f covers any two minterms of this subset. A function has many independent sets. The number of elements in a maximum independent set (MIS) of f is denoted by $\eta(f)$.

Property 2.2.1. Let f be a Boolean function. Then, $\eta(f) \leq \tau(f)$.

Proof. Each minterm in an MIS of f should be covered by a different prime implicant in an MSOP of f. *Q. E. D.*

Function f, such that $\eta(f) = \tau(f)$, is *orthodox* [18]. The orthodox functions include all unate, parity, and symmetric functions, as well as all functions depending on three or fewer variables, and all the functions with MSOPs composed of the essential primes. Also, about 98% of benchmark functions are orthodox [18]. It can be shown that this result is close to the observation in [9] that graphs appearing in the practical applications are 1-perfect.

Example 1. Fig. 1 shows a non-orthodox function of four variables. For this function, $\tau(f) = 5$ and $\eta(f) = 4$.

Theorem 2.2.1. [18] Let $f(X)$ and $g(Y)$ be disjoint-support orthodox functions. Then, $\tau(f \wedge g) = \tau(f) \cdot \tau(g)$.

Example 2. Let $f(x_1, x_2, x_3) = x_1 \vee x_2 \vee x_3$ and $g(y_1, y_2, y_3) = y_1 \vee y_2 \vee y_3$. Then, $f(x_1, x_2, x_3) \wedge g(y_1, y_2, y_3) = (x_1 \vee x_2 \vee x_3)(y_1 \vee y_2 \vee y_3) = x_1y_1 \vee x_2y_1 \vee x_3y_1 \vee x_2y_1 \vee x_2y_2 \vee x_2y_3 \vee x_3y_1 \vee x_3y_2 \vee x_3y_3$. For an SOP for $f \wedge g$ obtained by the distributive law, it is true that $\tau(f \wedge g) \leq \tau(f) \cdot \tau(g)$. The above theorem claims that the SOP is minimum. In other words, $\tau(f \wedge g) = \tau(f) \cdot \tau(g) = 9$.

This theorem, repeated below as Theorem 3.2.2, is the foundation of the results reported in this paper. Note that the theorem does not hold if f and g are disjoint-support but not orthodox. The result given in [18] shows that if f and g are defined as in Fig. 1, the equality $\tau(f \wedge g) = \tau(f) \cdot \tau(g)$ does not hold because $\tau(f \wedge g) = 24$ while $\tau(f) \cdot \tau(g) = 25$.

ab \ *cd*	00	01	11	10
00	0	1	0	1
01	0	1	1	1
11	1	1	1	1
10	1	0	0	0

Figure 1. A four-variable non-orthodox function.

2.3 Incompletely Specified Functions

The approach to SOP minimization developed in this paper is applicable to completely specified Boolean functions. However, the proof of one result (Theorem 3.2.5) requires the consideration of incompletely specified functions.

The *incompletely specified function* is $f(X)$: $B^n \rightarrow \{0,1,-\}$. We present such functions by their on-set and their don't-care set. The concept of support of incompletely specified functions is non-trivial because some variables, on which these functions depend, are potentially *vacuous* and can be removed without changing the function. In this paper, we use the concept of disjoint-support functions. Therefore, we define the *support* of an incompletely specified function as the largest set of variables, on which the function can depend. Two incompletely specified functions have *disjoint supports* if their supports have empty intersection. This definition is natural for the treelike decompositions of the type provided by disjoint-support decomposition.

Other definitions for incompletely specified functions are similar to the case of completely specified functions with the following difference: a *prime implicant* has non-empty intersection with the on-set. An MIS and the orthodox functions are defined similarly.

2.4 Disjoint Support Decomposition

Applying decomposition to a logic function results in a network of smaller subfunctions. *Disjoint support decomposition* (DSD), if it exists, produces networks, in which all subfunctions have single outputs and disjoint supports. DSD has *the finest granularity*, if the subfunctions cannot be further decomposed using DSD.

In this work, we rely on the fact that, for a completely specified Boolean function, DSD of the finest granularity is canonical [1]. It means that there exists a unique network (up to complementation of the subfunctions) with the property that none of its blocks can be further decomposed using DSD.

Even though DSD has been studied since 1950's [1], efficient algorithms to detect it have been discovered only recently [2][12]. In this paper, we use the fast algorithm [2] with improvements [12]. This algorithm computes the DSD network directly from the BDD representation of the function. The runtime to compute DSD, or to show that DSD does not exist, is negligible compared to the runtime of SOP minimization.

3. DECOMPOSITION THEORY

This section presents the theoretical foundations of the divide-and-conquer strategy to SOP minimization. The properties are based on the distance between minterms, DSD, and unateness.

3.1 Separate Minimization Using Distance

Theorem 3.1.1[20]. Let $f(X)$ and $g(X)$ be two functions such that the distance between every minterm of f and every minterm of g is two or more. Then, $\tau(f \vee g) = \tau(f) + \tau(g)$.

Example 3. Consider a function shown in Fig. 2 (left). The minterms of this function can be divided into two sets: $\{\bar{a}bcd, \bar{a}bc\bar{d}, abc\bar{d}\}$ and $\{ab\bar{c}d, a\bar{b}\bar{c}d, a\bar{b}cd\}$. The distance between the minterms in the sets is 2 or more. By Theorem 3.1.1, the functions represented by the sets of minterms can be minimized independently to produce an MSOP $\bar{a}bc \vee bc\bar{d} \vee a\bar{c}d \vee a\bar{b}d$.

$ab \setminus cd$	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	1	0	1
10	0	1	1	0

$ab \setminus cd$	00	01	11	10
00	0	1	0	0
01	1	1	0	0
11	1	1	0	1
10	0	1	1	1

Figure 2. Functions for Examples 3 and 4.

Theorem 3.1.2. Let $f(X)$ be a function such that $f(X) = \bar{x}\, g_1(X_1) \vee xyg_2(X_2)$, $x, y \in X$ and $x, y \notin X_1, X_2$. Then, $\tau(f) = \tau(g_1) + \tau(g_2)$.

Example 4. Consider function f shown in Fig. 2 (right). $f(X) = \bar{c}\, g_1(X_1) + cag_2(X_2)$, where $g_1(X_1) = b \vee d$ and $g_2(X_2) = \bar{b} \vee \bar{d}$. By Theorem 3.1.2, $\tau(f) = \tau(g_1) + \tau(g_2) = 2 + 2 = 4$.

3.2 Separate Minimization Using DSD

3.2.1. OR of Disjoint-Support Functions

Theorem 3.2.1. [18] Let $f(X)$ and $g(Y)$ be functions with disjoint supports, each not identically 1. Then, $\tau(f \vee g) = \tau(f) + \tau(g)$.

Example 5. Consider function h shown in Fig. 3 (left). $h = f \vee g$, where $f(X) = a \vee b$ and $g(Y) = c \vee d$. In this case, $\tau(f) = 2$, $\tau(g) = 2$. By Theorem 3.2.1, $\tau(h) = \tau(f \vee g) = 2 + 2 = 4$.

$ab \setminus cd$	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$ab \setminus cd$	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	0	1	1	1
10	1	0	0	0

Figure 3. Functions for Examples 5 and 6.

3.2.2. AND of Disjoint-Support Functions

Theorem 3.2.2. [18] Let $f(X)$ and $g(Y)$ be disjoint-support orthodox functions. Then, $\tau(f \wedge g) = \tau(f) \cdot \tau(g)$.

The above theorem shows that, in this case, an MSOP of $f \wedge g$ can be derived by applying distributive law to the MSOPs of f and g.

It can be observed that the results of Subsection 3.2.2 also hold for incompletely specified functions, as defined in Subsection 2.3.

3.2.3. EXOR of Disjoint-Support Functions

Theorem 3.2.3. Let $f(X)$ and $g(Y)$ be functions with disjoint supports. Let f and g, as well as their complements, $\bar{f}$ and $\bar{g}$, be orthodox. Then, $\tau(f \oplus g) = \tau(f) \cdot \tau(\bar{g}) + \tau(\bar{f}) \cdot \tau(g)$.

Example 6. Consider function h shown in Fig. 3 (right). $h = f \oplus g$, where $f = a \oplus b$ and $g = c \vee d$. $\tau(f) = 2$, $\tau(\bar{f}) = 2$, $\tau(g) = 2$, $\tau(\bar{g}) = 1$. Theorem 3.2.3 gives $\tau(h) = \tau(f \oplus g) = 2 \cdot 1 + 2 \cdot 2 = 6$.

3.2.4. MUX of Disjoint-Support Functions

Theorem 3.2.4. Let $f = (\bar{h} \wedge g_0) \vee (h \wedge g_1)$, where $h(X)$, $g_0(Y)$, and $g_1(Z)$ are functions with disjoint supports. Let g_0 and g_1 be not equal to constant 1. Let h, $\bar{h}$, g_0 and g_1 be orthodox. Then, $\tau(f) = \tau(\bar{h}) \cdot \tau(g_0) + \tau(h) \cdot \tau(g_1)$.

Example 7. Consider function f in Fig. 4. $f = (\bar{h} \wedge g_0) \vee (h \wedge g_1)$, where $h(c) = c$, $g_0(a, b) = a \oplus b$, and $g_1(d, e) = d \vee e$. Theorem 3.2.4 gives $\tau(f) = \tau(\bar{h}) \cdot \tau(g_0) + \tau(h) \cdot \tau(g_1) = 1 \cdot 2 + 1 \cdot 2 = 4$.

$ab \setminus cde$	000	001	011	010	110	111	101	100
00	0	0	0	0	1	1	1	0
01	1	1	1	1	1	1	1	0
11	0	0	0	0	1	1	1	0
10	1	1	1	1	1	1	1	0

Figure 4. Function for Examples 7.

3.2.5. Unate Composition of Disjoint-Support Functions

In this subsection, without the loss of generality, we consider the functions having one or more positive variables. The same properties work for the case of negative variables, by complementation of the variables and the corresponding decomposition subfunctions.

Theorem 3.2.5. Let function $g(x, Y)$, $x \notin Y$, be positive in x. Let an MSOP of g be $G = xG_1 \vee G_2$, where G_1 and G_2 do not depend on x. Let G_1 and G_2 be SOPs of g_1 and g_2, respectively. Let g^x be an incompletely specified function with the on-set $g_1 \wedge \bar{g}_2$ and the don't-care set g_2. Let both g^x and g_2 be orthodox. Let function $f(Z, Y)$ be represented as $g(h(Z), Y)$, where h is an orthodox function, and H is an MSOP of h. Then, $\tau(f) = \tau(h) \cdot \tau(g_1) + \tau(g_2)$. An MSOP of f can be derived by applying the distributive law to the expression $(H \wedge G_1) \vee G_2$.

$ab \setminus cd$	00	01	11	10
00	0	1	1	0
01	0	1	1	1
11	0	1	1	0
10	0	1	1	1

$ab \setminus cd$	00	01	11	10
00	1	0	1	1
01	1	1	0	0
11	1	0	1	1
10	1	1	0	0

Figure 5. Functions for Examples 8 and 9.

Example 8. Consider function f in Fig. 5 (left). $f = g(h(a, b), c, d)$, where $h(a, b) = a \oplus b$, and $g(x, c, d)$ has an MSOP $G = xc \vee d$ with $g_1 = c$ and $g_2 = d$. Theorem 3.2.5 gives $\tau(f) = \tau(h) \cdot \tau(g_1) + \tau(g_2) = 2 \cdot 1 + 1 = 3$. An MSOP of f is derived by applying the distributing law as follows: $((a \oplus b) \wedge c) \vee d = a\bar{b}c \vee \bar{a}bc \vee d$.

The following example shows that, in Theorem 3.2.5, function g must be positive in x.

Example 9. Consider function f shown in Fig. 5 (right). Function f can be represented as $g(h(a, b), c, d)$, where $h(a, b) = a \oplus b$ and $g(x, c, d)$ has an MSOP $G = \bar{x}c \vee \bar{x}\bar{d} \vee x\bar{c}$. Note that g is not positive in x. Applying the distributing law to G yields the expression $(\bar{a}\bar{b} \vee ab)c \vee (\bar{a}\bar{b} \vee ab)\bar{d} \vee (a\bar{b} \vee \bar{a}b)\bar{c}$ with 6 products. However, $\tau(f) = 5$ and an MSOP of f is $\bar{c}\bar{d} \vee a\bar{b}\bar{c} \vee \bar{a}b\bar{c} \vee \bar{a}\bar{b}c \vee \bar{a}\bar{b}c$.

It is possible to generalize Theorem 3.2.5 for function g, which has several positive variables.

Essentially, the simple version of Theorem 3.2.5 states that, to obtain an MSOP of $f(Z, Y) = g(h(Z), Y)$, we need (a) to show that g is positive in x, (b) to minimize g and h, and (c) to show that functions g^x and g_2, as defined in Theorem 3.2.4, are orthodox. This task is typically easier than minimizing an SOP of f in the brute-force way, because the number of products in f can be much larger than that in g and h. Section 4 show that, for the majority of practical functions, proving a function to be orthodox is easy. Furthermore, [18] shows that about 98% of benchmark functions (and their subfunctions) are orthodox.

3.3 Separate Minimization Using Unateness

Theorem 3.3.1. Let function $g(x, Y)$, $x \notin Y$, be positive in x. Let $g_2(Y) = g(0, Y) \wedge g(1, Y)$ and g_1 be the incompletely specified function with the on-set $g(1, Y) \wedge \bar{g}_2(Y)$ and the don't-care set $g_2(Y)$. Let G_1 and G_2 be MSOPs of g_1 and g_2, respectively. Then, $\tau(f) = \tau(g_1) + \tau(g_2)$ and an MSOP for g can be derived by applying the distributing law to the expression $(x \wedge G_1) \vee G_2$.

3.4 Separate Minimization Using DSD

A function $f(Z,Y)$ has a disjoint-support decomposition (DSD) if f can be written as $f(Z,Y) = g(h(Z),Y)$. Suppose that MSOPs for $h(Z)$ and $g(x,Y)$ be H and G, respectively. We can derive an SOP for $f(Z,Y)$ from H and G using de Morgan's law and the distributive law. However, the SOP derived by this method is not always minimum. For example, when $g(x, Y) = x * h(Y)$ and $x = h(Z)$, where h is the function defined in Fig. 1, the SOP generated by this method contains 25 products, but $\tau(f(Z,Y)) = 24$.

3.5 Case Study: Benchmark Function *t481*

In some cases, an MSOP of a function can be derived from the theory presented in Sections 3.1-3.3, without using the classical minimization algorithms.

Consider the benchmark function *t481*, which has 16 inputs and the DSD shown in Fig. 6 [21]. The DSD is computed from the BDD of *t481* by the algorithm [2][12] in less than 0.001 sec. Because every logic level in the DSD structure is composed of the same gates, we consider one gate per level. Let the gates *A, B, C*, and *D* produce the functions f_A, f_B, f_C, and f_D, respectively.

Gate A is an AND with one complemented input. The number of products in an MSOP of f_A and its complement are equal to 1 and 2, respectively. That is, $\tau(f_A) = 1$ and $\tau(\bar{f}_A) = 2$.

Function f_B of gate B is an EXOR of two disjoint-support orthodox functions of type f_A. By Theorem 3.2.3, $\tau(f_B) = \tau(f_A) \cdot \tau(\bar{f}_A) + \tau(\bar{f}_A) \cdot \tau(f_A) = 1 \cdot 2 + 2 \cdot 1 = 4$. Similarly, $\tau(\bar{f}_B) = \tau(\bar{f}_A) \cdot \tau(\bar{f}_A) + \tau(f_A) \cdot \tau(f_A) = 2 \cdot 2 + 1 \cdot 1 = 5$.

Function f_C of gate C is an AND of two disjoint-support orthodox functions of type f_B in different polarities. By Theorem 3.2.2, $\tau(f_C) = \tau(f_B) \cdot \tau(\bar{f}_B) = 4 \cdot 5 = 20$. By Theorem 3.2.1, $\tau(\bar{f}_C) = \tau(f_B) + \tau(\bar{f}_B) = 4 + 5 = 9$.

Function f_D is an EXOR of two disjoint-support orthodox functions of type f_C. By Theorem 3.2.3, $\tau(f_D) = \tau(f_C) \cdot \tau(\bar{f}_C) + \tau(\bar{f}_C) \cdot \tau(f_C) = 20 \cdot 9 + 9 \cdot 20 = 360$. Similarly, $\tau(\bar{f}_D) = \tau(\bar{f}_C) \cdot \tau(\bar{f}_C) + \tau(f_C) \cdot \tau(f_C) = 9 \cdot 9 + 20 \cdot 20 = 481$.

Because of the inverter at the output of gate D, the number of products in an MSOP of *t481* is 481.

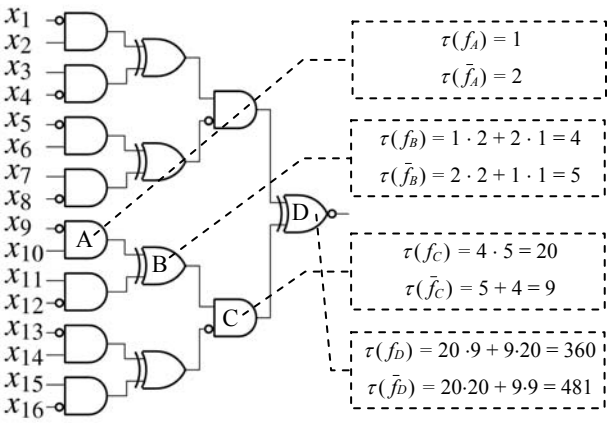

Figure 6. DSD Network for Benchmark Function *t481*.

4. MUSASHI ALGORITHM

The proposed algorithm, MUSASHI, works as a preprocessor to a general-purpose SOP minimizer. It splits a large SOP minimization problem into a number of simpler subproblems. The subproblems may be trivial or may require classical minimization methods to be applied. Finally, the minimum SOP is assembled from the partial solutions.

4.1 Outline of MUSASHI

The high-level pseudo-code of the algorithm is shown in Fig. 7. The input to MUSASHI is a single-output completely specified function and the output is an MSOP of this function. The result of minimization can be exact or heuristic, depending on the type of minimization used for the subfunctions.

First, the canonical DSD for the function is computed. Next, the DSD network is recursively collapsed starting from the primary output node. This procedure is based on Theorem 3.2.5. The non-decomposable functions are partitioned using Theorems 3.1.1 and 3.1.2. At this point, MUSASHI recursively applies the DSD computation to the partitions (the pseudo-code does not show this step). Next, the nodes of the network are minimized using the general-purpose minimizer. Finally, the SOP is constructed from the SOPs for the function at each node, using the distributive law.

```
MUSASHI ( function )
{
   network = PerformDisjointSupportDecomposition( function );
   SelectivelyCollapseDSDNetwork( PrimaryOutput( network ) );
   PartitionNonDecomposableFunctionsUsingDistance( network );
   MinimizeSubfunctionsIndependently( network );
   SOP = ConstructSOPUsingDistributiveLaw( network );
   return SOP;
}
```

Figure 7. MUSASHI Pseudo-Code.

4.2 Detection of Orthodox Functions

To prove that a function is orthodox, it is enough to show that the number of products in an MSOP of this function is equal to the size of an MIS. Both problems have exponential complexity. Therefore, in the general case, proving that a function is orthodox is as difficult as performing the exact SOP minimization.

It has been shown by a computer program [18] that, for the random functions, the percentage of the orthodox functions is approaching 0 when the number of support variables increases. However, the situation is different for the functions from the practical applications. It was shown that 98% of the benchmark functions are orthodox. Moreover, proving such functions to be orthodox is easy, using the following necessary condition.

Property 4.3.2. Function *f* is orthodox, if the number of products in an SOP of *f*, computed by a heuristic algorithm, is equal to the size of an independent set of *f,* computed by a heuristic algorithm.

In our implementation, this simple method detects the orthodox functions in approximately 95% of cases. In the remaining 5% of cases, either (1) the function is not orthodox, or (2) the function is orthodox but one of the heuristic algorithms (either SOP or independent set computation) has failed to find an exact solution. In the case of (2), it may be possible to improve the quality of the heuristic algorithms and answer the question whether the function is orthodox. However, the additional computational effort can substantially increase the runtime. Because we are interested in speeding up the computation, we ignore such cases and, instead, perform SOP minimization using the traditional methods.

5. EXPERIMENTAL RESULTS

MUSASHI is implemented using the BDD package CUDD [23] with extensions found in EXTRA library [15]. The program has been tested on a 2Ghz Pentium 4 PC with 256Mb RAM.

We analyzed ESPRESSO benchmarks intending to use them in our experiments. Although most the single-output functions in this benchmark set have abundant DSDs, there was no significant runtime improvement because the SOPs are small ($\tau(f) \leq 50$) and ESPRESSO typically minimizes them in less than 0.1 sec.

To demonstrate the effectiveness of MUSASHI for "large-scale" SOP minimization, we considered single-output functions of the combinational and sequential MCNC benchmarks that have large support sizes and non-trivial DSDs. The results are divided into three categories and presented in Table 1 as follows: (1) comparison with ESPRESSO-heuristic, (2) comparison with ESPRESSO-exact, and (3) results for the functions, which both options of ESPRESSO could not finish in 5 minutes.

Table 1. MUSASHI vs. Conventional Minimization.

Circuit	Out	Ins	BDD	Primes	Offset	Onset	Max	Part	MUS	Espr
(1) Comparison with ESPRESSO-heuristic										
C880	17	29	81	542	3428	198	22	196	0.06	0.17
des	56	18	52	3137	3143	1025	12	528	0.20	0.23
i10	37	50	156	24308	6722	1579	45	1034	1.77	4.24
i10	119	48	175	12676	1261	3628	30	413	0.36	19.55
i10	188	31	51	1033	65544	1033	8	7	0.01	32.39
pair	4	51	74	124	68904	50	13	15	0.01	0.58
rot	16	49	275	3407	7113	786	31	249	0.28	16.13
rot	84	43	218	4215	2456	947	36	540	0.78	8.55
rot	86	34	55	904	47	544	22	56	0.02	1.84
s1423	52	51	219	50162	10523	3108	44	831	1.50	27.36
s1423	71	36	236	81925	70496	676	31	409	0.41	0.78
s15850	482	34	59	21408	21409	1664	22	27	0.02	1.25
s5378	185	24	37	1044	8352	272	9	10	0.01	0.25
Total		498					325		5.43	113.32
(2) Comparison with ESPRESSO-exact										
C5315	43	22	163	1296	44	950	11	25	0.01	0.67
C5315	44	26	215	13797	74	4084	20	1021	3.50	104.53
cordic	1	23	36	1539	215	771	7	4	0.01	0.36
des	100	19	47	9479	9479	1764	12	864	0.39	1.99
i10	29	49	98	5501	2028	1311	35	829	3.00	10.45
i10	78	40	139	3847	1136	477	32	185	0.53	2.66
i10	84	41	271	11403	4334	741	37	481	7.91	31.06
i2	0	201	206	4439	384	214	6	4	0.01	7.75
rot	10	45	216	2966	2870	1260	31	265	0.73	24.69
rot	16	49	275	3407	7113	786	31	249	0.31	8.91
rot	84	43	218	4215	2456	947	36	540	1.05	21.94
s1423	55	51	124	15574	4100	875	44	344	0.20	26.92
s15850	389	48	69	3454	13087	211	22	25	0.02	1.97
s15850	592	39	61	3898	896	40	26	32	0.03	2.78
s9234	44	40	49	15360	989	34	17	17	0.02	42.00
s9234	187	45	53	3942	689	36	15	15	0.01	3.92
too_lar	2	35	200	1914	280	523	30	504	1.56	2.72
Total		816					412		19.29	295.32
(3) Functions, which ESPRESSO cannot solve in 5 minutes										
C2670	135	42	55	788224	$3\cdot10^{10}$	1792	6	4	0.01	-
C2670	138	119	205	$1\cdot10^{8}$	$3\cdot10^{10}$	1.3e08	6	4	0.01	-
C5315	68	20	21	131840	131840	768	0	0	0.01	-
C7552	69	65	106	$3\cdot10^{6}$	$7\cdot10^{16}$	3008	21	836	0.31	-
i10	13	31	53	65543	3131	65542	13	7	0.01	-
i10	45	37	55	15452	$1\cdot10^{6}$	531	13	16	0.01	-
s15850	2	140	177	$1\cdot10^{33}$	$1\cdot10^{10}$	1638	26	31	0.02	-
s15850	240	117	133	$4\cdot10^{8}$	$6\cdot10^{11}$	141	20	24	0.01	-
s15850	472	52	78	481682	481682	17682	22	27	0.02	-
s9234	20	36	58	67734	63630	20498	17	40	0.01	-
Total		659					144		0.42	

The following notation is used in Table 1. “Circuit” contains the name of the multi-output benchmark functions. “Out” is the zero-based number of the output used in our experiment. “Ins” is the number of inputs in the true support of this output. “BDD” is the number of BDD nodes after reordering. The following columns characterize the SOP: “Primes” is the number of all primes; “Offset” is the number of products in the ISOP of the offset, computed using algorithm [14]; “Onset” is the number of products in the SOP generated by MUSASHI. This result was verified by running ESPRESSO. In all cases when ESPRESSO completed, it returned the same number of products as MUSASHI.

Columns “Max” and “Part” contain the support size and the MSOP size of the largest node in the DSD network, which had to be minimized using our approach. Finally, “MUS” and “Espr” show the runtime, in seconds, of our algorithm and ESPRESSO, respectively. Table 1 shows that the efficiency of the divide-and-conquer strategy is proportional to the reduction in the support of the components to be minimized separately (“Max”), compared to the support size of the original benchmark function (“Ins”).

The runtime of MUSASHI reported in column “MUS” includes that for several procedures but the details are omitted in Table 1 due to the page limitation. On average, SOP minimization and the MIS computation used to detect the orthodox functions take about 90% and 8% of runtime, respectively. The remaining 2% of runtime are taken by all other procedures: DSD computation, selective network collapsing, and deriving the resulting SOPs.

It might be possible to improve the performance of ESPRESSO in our experiments by computing the reduced offset in the heuristic minimization [11] and using different approaches to the exact minimization [13][7][8]. However, this observation does not diminish the value of the proposed divide-and-conquer strategy. Given a more efficient SOP minimizer and larger benchmark functions, our approach will allow for a *comparable* improvement in performance, which is achieved by eliminating the need for time-consuming SOP minimization of large functions and minimizing smaller subfunctions instead.

6. CONCLUSIONS

In this paper, we explore a number of conditions when an exact minimum SOP can be found by a divide-and-conquer strategy. In particular, we rely on a class of decomposable orthodox functions, for which SOP minimization can be derived from the decomposition tree of the function.

The theory developed in the paper is useful for practical applications because the majority of benchmark functions have disjoint-support decompositions with subfunctions that satisfy the following conditions: (1) they are simpler than the original functions, and (2) they are *orthodox* and unate in some variables. The implementation of the algorithm is efficient because (*a*) a fast BDD-based algorithm to detect all disjoint-support decomposition is available [2][12], and (*b*) in most cases, detection of orthodox functions can be performed in a time polynomial in the SOP size.

The experimental results show that MUSASHI , when used as a preprocessing step to the general-purpose SOP minimizer, can speed up the computation by several orders of magnitude. Using MUSASHI, some large benchmark functions are minimized exactly for the first time.

7. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support from Kyushu Institute of Technology under the 75th Commemoration Fund Program for Foreign Researchers. This research is partially supported by the Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS), and a grant from the Takeda Foundation. The first author has been partially supported by a research grant from Intel Corporation. The authors thank Prof. Jon Butler for helpful discussions.

8. REFERENCES

[1] R. L. Ashenhurst, “The decomposition of switching functions”. *Computation Lab*, Harvard University, 1959, Vol. 29, pp. 74-116.

[2] V. Bertacco and M. Damiani, "Disjunctive decomposition of logic functions," *Proc. ICCAD ‘97*, pp. 78-82.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli*, Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Dordrecht, 1984.

[4] R. K. Brayton and C. McMullen, “The decomposition and factorization of Boolean expressions,” *Proc. ISCAS ‘82*, pp. 29-54.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comp.*, Vol. C-35, No. 8 (August, 1986), pp. 677-691.

[6] M.J. Ciesielski, S. Yang, and M.A. Perkowski, "Multiple-valued minimization based on graph coloring," *Proc. ICCD '89*, pp.262-265.

[7] O. Coudert and J. C. Madre, “Towards a symbolic logic minimization algorithm”, *Proc. VLSI Design*, January 1993.

[8] O. Coudert, “Two-level logic minimization: An overview”, *Integration*, 17-2, pp. 97-140, Oct. 1994.

[9] O. Coudert, “Exact coloring of real-life graphs is easy”, *Proc. DAC ’97*, pp. 121-126.

[10] J. Jacob and A. Mishchenko, “Unate decomposition of Boolean functions”, *Proc. IWLS ’01*, pp. 66-71.

[11] A. A. Malik, R. Brayton, A. R. Newton and A. Sangiovanni-Vincentelli, “Reduced offsets for two-level multi-valued logic minimization”, *Proc. DAC’ 90*, pp. 290-296.

[12] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition", *Proc. SASIMI '98*, pp. 44-50.

[13] P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli, "Espresso-Signature: A new exact minimizer for logic functions," *Proc. DAC ’93*, pp. 618-624.

[14] S. Minato, “Fast generation of irredundant sum-of-products forms from binary decision diagrams. *Proc. SASIMI'92*, pp. 64-73.

[15] A. Mishchenko, *EXTRA Library of DD procedures*. http://www.ee.pdx.edu/~alanmi/research/ extra.htm

[16] R. L. Rudell and A. Sangiovanni-Vincentelli, “Multiple-valued minimization for PLA optimization”, *IEEE Trans. CAD*, Vol. 6(5), pp. 727-750, Sep. 1987.

[17] T. Sasao, "An algorithm to derive the complement of a binary function with multiple-valued inputs," *IEEE Trans. Comp*. Vol. C-34, No. 2, pp. 131-140, Feb. 1985.

[18] T. Sasao and J. T. Butler, "On the minimization of SOPs for bi-decomposable functions", *Proc. ASP-DAC '01*, pp.219-224.

[19] T. Sasao and J. T. Butler, "Worst and best irredundant sum-of-products expressions", *IEEE Trans. Comp*, Vol. 50, No. 9, Sept. 2001, pp. 935-948.

[20] T. Sasao, S. J. Hong, and R. K. Brayton, “Minimization of PLA’s by decomposition”, *Unpublished technical report,* 1984.

[21] T. Sasao and M. Matsuura, "DECOMPOS: An integrated system for functional decomposition," *Proc. IWLS ’98*, pp. 471-477.

[22] E. Sentovich, et al, “SIS: A system for sequential circuit synthesis”, *Tech. Rep. UCB/ERI, M92/41, ERL*, Dept. of EECS, Univ. of California, Berkeley, 1992.

[23] F. Somenzi, BDD package “CUDD v. 2.3.0.” http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html

[24] http://www.lsi-cad.com/dac2003/appendix.pdf.