# An Approach to Disjoint-Support Decomposition of Logic Functions

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

## Abstract

*This paper describes a new approach to disjoint-support decomposition. Its advantage over Bertacco-Damiani [1] and Minato-DeMicheli [2] is that it is relatively simple and easy to implement. The advantage over Sasao-Matsuura [3] is that the run time is reasonable (less than 1 sec for functions with 20-25 inputs). It has not been proved that this method determines all disjoin-support decompositions. However, it has found all known disjoint-support decompositions when tested on artificially generated examples as well as on a selection of PLA and BLIF MCNC benchmarks.*

## 1 Introduction

There are several approaches to disjoin-support decomposition. Approach [1] uses the BDD structure as a guide for decomposition. In this approach, disjoint-support decomposition for a function is derived by first deriving it for the cofactors of the function and then comparing the resulting decomposition lists. Approach [2] uses the properties of recursive computation of Irredundant Sum-of-Products to derive factored forms, from which disjoint decomposition can be determined. Approach [3] uses Jacobian introduced in [4] to find candidate variable sets for disjoint-support decomposition.

The proposed approach is similar to [3] in that it uses the specialized formula to determine the likely bound sets. However, the formula used is essentially different from Jacobian and computationally is less expansive. Additional speed-up has been achieved due to efficient implementation, in particular, using BDDs to store the intermediate results and test whether the decomposition with the given bound set exists.

The following sections contain the outline of the decomposition algorithm.

## 2 Definitions

Given a Boolean function F: $B^n \rightarrow B$, where B = {0,1}, the *negative (positive) cofactor* of F with respect to (w.r.t.) variable x is the Boolean function $F_0$ ($F_1$) derived by substituting into F instead of x the value 0 (1). Similarly, it is possible to define cofactors w.r.t. a number of variables, in particular, two variables. For example, assuming that x and y are variables of the function, $F_{01}$ is derived by substituting 0 instead of x and 1 instead of y.

In this paper, the function is said to satisfy the *decomposition property* with variables {x,y} if the formula is true for the function: $F_{00}$ & $F_{11}$ = $F_{01}$ & $F_{10}$. Notice that in the following sections this formula is considered not for the function but for the Boolean difference of the function.

The Boolean difference of function F w.r.t. a variable x is Exclusive OR of its two cofactors w.r.t. to x: $D_{F,x} = F_0 \oplus F_1$.

Function F(X) is *disjoint-decomposable* with *bound set* $X_1$ = { $x_{i1}$,$x_{i2}$,$x_{i3}$, …} and *free set* $X_2$ = { $x_{k1}$,$x_{k2}$,$x_{k3}$, …}, if $X_1$ and $X_2$ form a disjoint partition on variable set X, and F(X) can be represented as F(X) = H( G($X_1$), $X_2$ ), where G($X_1$) is a single-output Boolean function.

Function F(X) is *bi-decomposable* with AND-gate and variable sets $X_1$ = {$x_{i1}$,$x_{i2}$,$x_{i3}$,…} and $X_2$ = {$x_{k1}$,$x_{k2}$,$x_{k3}$,…}, if F can be represented as F(X) = A($X_1$) & B($X_2$),

## 3 Theory

***Theorem 1.*** *If the function is bi-decomposable with the AND-gate, it satisfies the decomposition property.*

The opposite is not true. The function may satisfy the property and yet have no disjoint decomposable. Consider an example: $F(a,b,c,d) = b\bar{c}\bar{d} + \bar{a}d + cd + \bar{b}c$ . This function is not AND-bi-decomposable with variable sets {a,b} and {c,d}, as can be verified by drawing its Karnaugh map, yet it satisfies decomposition property w.r.t. variables {a,b}.

***Theorem 2.*** *Let function F be disjoint-decomposable with bound set $X_1$ and free set $X_2$. Let $z \in X_1$, and a and b belong to different sets ($a \in X_1$, $b \in X_2$ or $a \in X_2$, $b \in X_1$), then the Boolean difference of F w.r.t. variable z satisfies the decomposition property with variables {a,b}.*

**Proof:** The theorem is proved by showing that if the function has disjoint-support decomposition with $X_1$ and $X_2$, then its Boolean difference w.r.t. a variable in the bound set is AND-bi-decomposable with a pair of variables, one of which belongs to the bound set and another belongs to the free set.

According to Theorem 1, the Boolean difference of this function satisfies the decomposition property.

Let us show that the Boolean difference w.r.t. a variable in the bound set is AND-bi-decomposable.

$F = F( G(z,a,...), b,...)$

$F_{z=0} = F( G(0,a,...), b,...)$

$F_{z=1} = F( G(1,a,...), b,...)$

Perform Shannon expansion w.r.t. G as an input of F.

$$F_{z=0} = F( 0, b, ...) \ \& \ \overline{G(0,a,...)} \oplus F( 1, b, ...) \ \& \ G(0,a,...)$$

$$F_{z=1} = F( 0, b, ...) \ \& \ \overline{G(1,a,...)} \oplus F( 1, b, ...) \ \& \ G(1,a,...)$$

Compute the Boolean difference of F w.r.t. z:

$$\begin{aligned} D_{F,z} &= F_{z=0} \oplus F_{z=1} \\ &= F( 0, b, ...) \ \& \ \overline{G(0,a,...)} \oplus F( 1, b, ...) \ \& \ G(0,a,...) \\ &\oplus F( 0, b, ...) \ \& \ \overline{G(1,a,...)} \oplus F( 1, b, ...) \ \& \ G(1,a,...) \\ &= F( 0, b, ...) \ \& \ [ \overline{G(0,a,...)} \oplus \overline{G(1,a,...)} ] \\ &\oplus F( 1, b, ...) \ \& \ [ G(0,a,...) \oplus G(1,a,...) ] \\ &= F( 0, b, ...) \ \& \ [ G(0,a,...) \oplus G(1,a,...) ] \\ &\oplus F( 1, b, ...) \ \& \ [ G(0,a,...) \oplus G(1,a,...) ] \\ &= [F( 0, b, ...) \oplus F( 1, b, ...)] \ \& \ [ G(0,a,...) \oplus G(1,a,...) ] \end{aligned}$$

It proves that the Boolean difference w.r.t. a variable in the bound set is indeed AND-bi-decomposable.

## 4 Main Algorithm

Disjoint-support decomposition is performed recursively. A single-output Boolean function is given to the algorithm. The algorithm tries to decompose the function by considering several types of bound sets. As soon as the decomposition is found, it is performed and two new functions are derived: $G(X_1)$ and $H(g,X_2)$. The decomposition is now called recursively for these two components.

The pseudo-code of the decomposition algorithm is given in Fig. 1.

```
DisjointDecompose( func F )
{
   varset V;
   if ( V = DecExists1( F ) )
       goto DECOMPOSE;
   if ( V = DecExists2( F ) )
       goto DECOMPOSE;
   if ( V = DecExists3( F ) )
       goto DECOMPOSE;

   for ( all variables z in support of F ) {
     while ( CandidateVarSetExists( F, z ) ) {
        if ( V = DecExists4( F ) )
           goto DECOMPOSE;
        if ( V = DecExists5( F ) )
           goto DECOMPOSE;
     }
   }

DECOMPOSE:
   if ( V != 0 ) {
      ( G, H ) = PerformDecomposition( F, V );
       DisjointDecompose ( G );
       DisjointDecompose ( H );
   }
}
```

Fig. 1. The outline of the decomposition algorithm.

Procedure DecExists1() tries to find disjoint-support decomposition with two variables in the bound set by cofactoring the function and counting the number of pairs of equal cofactors. For example, F = H( a & c, b,...).

Procedure DecExists2() tries to find decompositions with a two-input gate at the output of the function. For example, OR-disjoint-decomposition of this kind is F = a + G(b,...).

Procedure DecExists3() tries the difference of supports of F(X) and $D_{F,z}$ as a possible bound set for decomposition.

Procedure CandidateVarSetExistis() tries to find a likely bound set using the decomposition property. When the bound set is found, DecExists4() tries it as it is, while DecExists5() tries the same bound set with variable z added.

## 5 Deriving Candidate Bound Sets

The decomposability condition can be used to find candidate bound sets, which are next tested for disjoint-decomposition using a specialized BDD-based method.

To this end, the variables in the support of the function are considered one-by-one. The Boolean difference is derived for each variable and the decomposability condition is tested for each pair of variables in the support of the Boolean difference.

The result of testing the decomposability condition is represented as a matrix with as many rows and columns as there are variables in the support of the Boolean difference. If the decomposability condition is true for a pair of variables, the corresponding cell of the matrix is labeled by the cross; otherwise, it remains blank.

A typical matrix of this kind is shown in Fig. 2.

$$\begin{array}{c|cccccc|}
 & a & b & c & d & e & f \\
\hline
a & & & & & x & x \\
b & & & & & x & x \\
c & & & & & x & x \\
d & & & & & x & x \\
e & x & x & x & x & & \\
f & x & x & x & x & & \\
\hline
\end{array}$$

Fig. 1. A typical disjoint-support decomposition matrix.

The fact that the decomposability condition is true for variables {a,e} means, according to Theorem 2, that the function is can have disjoint decomposition with these variables belonging to different variable sets: {a} in the bound set and {e} in the free set, or vice versa. The same can be stated about other variable pairs, which have the cross in the corresponding cell. The general conclusion is that variables {a,b,c,d} should be tested as a candidate bound set for disjoint-decomposition.

If the function is not decomposable with this bound set, it may happen that the variable z, with respect to which the Boolean difference was computed, also belongs to the bound set, so next the algorithm will test the candidate bound set {z,a,b,c,d}.

By generalizing this approach it is possible to reduce the search for candidate bound sets to the computation of strongly connected components in the graph induced by the matrix.

## 6 Checking Disjoint Decomposition

A specialized BDD-based procedure has been developed for the purpose (see “bDecomp.c” in EXTRA library).

## 7 Conclusions

This method works remarkably well for many benchmarks. Its performance degrades when the number of inputs is more than 30. Advantages and disadvantages of the method are listed in the Abstract.

The algorithm can be improved in the following ways:

1) It can be modified by developing a heuristic strategy determining the order of testing bound set. In this way, the run time for decomposable benchmarks can be reduced because the decomposition will be found after a small number of trials-and-errors. For non-decomposable functions, however, the runtime will remain the same, because all candidate bound sets will have to be tested to “prove” non-decomposability.
2) There may be a way to trade the “provable quality” for “runtime” by considering only a part of all candidate bound sets and if the function is not decomposable on them, declaring it not decomposable at all.
3) A number of improvements can also be made to the BDD-based procedure, which tests the bound set and performs the decomposition if it exists.

## References

[1] V. Bertacco, M. Damiani. "The Disjunctive Decomposition of Logic Functions". *Proc. of ICCAD '97*, pp. 78-82.

[2] S. Minato, G. DeMicheli. "Finding All Simple Disjunctive Decompositions Using Irredundant Sum-of-Products Forms". *Proc. of ICCAD '98*, pp. 111-117.

[3] T.Sasao, M.Matsuura. “DECOMPOS: An Integrated System for Functional Decomposition”. *Proc. of IWLS ’98*, pp. 471-477.

[4] V. Y.-S. Shen, A.C. Mckellar, P.Weiner, “A fast algorithm for disjunctive decomposition of logic functions”. *IEEE Trans. Comps*. Vol. C-20, No. 3, pp. 304-309, March 1971.