

# Bi-decomposition of large Boolean functions using blocking edge graphs

Mihir Choudhury and Kartik Mohanram

Department of Electrical and Computer Engineering, Rice University, Houston

{mihir, kmram}@rice.edu

## Abstract

Bi-decomposition techniques have been known to significantly reduce area, delay, and power during logic synthesis since they can explore multi-level `and`, `or`, and `xor` decompositions in a scalable technology-independent manner. The complexity of bi-decomposition techniques is in achieving a good variable partition for the given logic function. State-of-the-art techniques use heuristics and/or brute-force enumeration for variable partitioning, which results in sub-optimal results and/or poor scalability with function complexity. This paper describes a fast, scalable algorithm for obtaining provably optimum variable partitions for bi-decomposition of Boolean functions by constructing an undirected graph called the blocking edge graph (BEG). To the best of our knowledge, this is the first algorithm that demonstrates a systematic approach to derive disjoint and overlapping variable partitions for bi-decomposition. Since a BEG has only one vertex per input, our technique scales to Boolean functions with hundreds of inputs. Results indicate that on average, BEG-based bi-decomposition reduces the number of logic levels (mapped delay) of 16 benchmark circuits by 60%, 34%, 45%, and 30% (20%, 19%, 16% and 20%) over the best results of state-of-the-art tools FBDD, SIS, ABC, and an industry-standard synthesizer, respectively.

## 1. Introduction

Decomposition of Boolean functions is a well-researched area with work that can be traced back to the 1950s. Decomposition is an important step during logic synthesis because it can significantly reduce the area, delay, and power of the final design [1, 2]. As designs increased in size and complexity, logic synthesis algorithms favored decomposition techniques based on local transformations to ensure scalability to large circuits. Although techniques based on local transformations are fast, they explore only a small space of alternatives and are also constrained by the structure of the initial netlist. In contrast, functional decomposition techniques are a class of decomposition techniques that recursively decompose a canonical representation of the given function into smaller sub-functions. Since they manipulate circuits at the functional level and do not depend on the starting netlist, they can explore a larger space of options and have been shown to significantly reduce the area, delay, and power of logic circuits.

Bi-decomposition, a form of functional decomposition wherein a function is recursively decomposed into two smaller functions, is an effective decomposition technique since it can be used to explore multi-level `and`, `or`, and `xor` decompositions. Bi-decomposition techniques rely on the ability to split the given logic function into two functions that depend on fewer input variables. Since the variable partition can significantly impact the quality of the decomposition, determining a good variable partition is not only the most

important, but also the most computationally intensive step during bi-decomposition. Although techniques for obtaining a variable partition using the structural properties of a binary decision diagram (BDD) — which is a canonical representation for the given logic function — have been proposed [3, 4], they are memory intensive and very sensitive to the variable order of the BDD used to extract the variable partition. Other bi-decomposition algorithms such as [5] obtain a variable partition using heuristics and/or brute-force enumeration of variable partitions, which compromises solution quality and/or offers poor scalability. Although the use of a SAT solver coupled with interpolation techniques [6] has been proposed to reduce the runtimes for variable partitioning, there is no control over the quality of the variable partition obtained using such approaches.

This paper describes a fast, scalable algorithm for obtaining provably optimum variable partitions for bi-decomposition of Boolean functions by constructing an undirected graph called the blocking edge graph (BEG). To the best of our knowledge, this is the first algorithm that demonstrates a systematic approach to derive disjoint and overlapping variable partitions for the bi-decomposition of a logic function. Given a logic function  $f$  with  $n$  inputs, we show that simple pairwise variable co-factoring information can be used to derive a necessary and sufficient condition for a pair of variables to occur in the same partition of a bi-decomposition. Based on this condition, a BEG is constructed to evaluate potential `and`, `or`, and `xor` bi-decompositions of the logic function. We show that a function is bi-decomposable *iff* the BEG for either an `and`, `or`, or `xor` bi-decomposition is not a complete graph. For bi-decomposable functions, we show that disjoint and overlapping variable partitions can be extracted by analyzing the vertex cuts of the BEG. We extract variable partitions from BEGs to minimize two commonly used metrics that have been known to produce circuits with a small area, delay, and power footprint: (i) the total number of variables in the partitions and (ii) the size of the largest partition. The variable partitions can be proved to be optimal with respect to both these metrics when they are obtained from the minimum vertex cuts of the BEG. Since a BEG has only one vertex per input of the function, our variable partitioning technique is scalable to functions with several hundred inputs. Further, the BEGs can also be used to identify functions that are not bi-decomposable. In such cases, we describe a decomposition technique based on relaxing the original function to make it bi-decomposable. Results indicate that on average, BEG-based bi-decomposition reduces the number of logic levels (mapped delay) of 16 benchmark circuits by 60%, 34%, 45%, and 30% (20%, 19%, 16% and 20%) over the best results of state-of-the-art tools FBDD, SIS, ABC, and an industry-standard synthesizer, respectively.

This paper is organized as follows. Section 2 describes variable partitioning based on BEGs. Section 3 describes function bi-decomposition based on the variable partition identified using BEGs. Section 4 presents results and Section 5 is a conclusion.

## 2. Bi-decomposition using blocking edge graphs

A function  $f$  of  $n$  variables is called *bi-decomposable* if it can be decomposed into two logic functions, each of which depends on less than  $n$  variables. The two smaller decomposed functions are combined using a two-input logic function. All two-input functions can be reduced to and, or, and xor operations upto complementation of inputs/output. Since any circuit with two-input gates can be reduced to a circuit with and, or, and xor gates by bubbling inverters down to the primary inputs, bi-decomposition techniques consider only and, or, and xor bi-decompositions of a logic function. Since and and or are dual operations, we obtain an and bi-decomposition for  $f$  from an or bi-decomposition of  $f$  by swapping the off-set and the on-set of  $f$  in this paper.

Bi-decomposition techniques obtain smaller decomposed functions by first obtaining a variable partition of the variable set,  $V$ , of the given function  $f$ . A variable partition consists of two variable sets  $V_1$  and  $V_2$ , such that  $|V_1| < |V|$  and  $|V_2| < |V|$ . A variable partition is disjoint if  $V_1 \cap V_2 = \phi$ , otherwise the variable partition is overlapping. Variable partitions depend on the type of bi-decomposition — and, or, or xor — that we seek for the given logic function  $f$ . Hence, the most important and computationally intensive step during bi-decomposition involves determining the kind of decomposition and the variable partition for the given logic function.

Our technique uses undirected graphs called *blocking edge graphs (BEGs)* to extract variable partitions for and, or, and xor bi-decompositions of a logic function. In this section, we first describe a necessary and sufficient condition, referred to as the *blocking condition*, for a pair of variables to be in the same variable partition of an and, or, or xor bi-decomposition of  $f$ . We then describe the steps for constructing separate BEGs for and, or, and xor bi-decompositions of a logic function based on the blocking condition. Finally, we show how variable partitions for and, or, and xor bi-decompositions can be extracted from the BEGs.

### 2.1 Blocking condition

For a pair of input variables,  $\{i, j\}$ , given a 0/1 assignment  $c$  of the variables in  $V \setminus \{i, j\}$ , the K-map of  $f$  can be restricted to a  $2 \times 2$  square covering the four cells  $c \cdot \bar{i}\bar{j}$ ,  $c \cdot i\bar{j}$ ,  $c \cdot \bar{i}j$ , and  $c \cdot ij$ . There are  $2^{n-2} \cdot 2 \times 2$  squares associated with the variable pair  $\{i, j\}$ , one for each 0/1 assignment of variables in  $V \setminus \{i, j\}$ . We classify the  $2 \times 2$  squares into 6 types based on the value of  $f$  in the four cells (see Fig. 1): (i) zero square with all four cells assigned 0, (ii) and square with three cells assigned 0, (iii) literal square with two adjacent cells assigned to 0, (iv) xor square with any two non-adjacent cells assigned to 0, (v) or square with three cells assigned to 1, and (vi) one square with all four cells assigned to 1. Note that although Fig. 1 shows only one  $2 \times 2$  square for each type, there are 4 different and squares, 4 different or squares, 4 different literal squares, and 2 different xor squares for a total of 16  $2 \times 2$  squares.

Given a logic function,  $f$ , of  $n$  variables and input variable set,  $V$  ( $|V| = n$ ), a variable partition,  $V_1$  and  $V_2$ , of  $f$  separates a pair of variables  $\{i, j\}$  if  $i \in V_1 \setminus V_2$  and  $j \in V_2 \setminus V_1$ . Based on the types of  $2 \times 2$  squares in  $f$  associated with a pair of variables  $\{i, j\}$ , we now describe the blocking condition for  $\{i, j\}$  in an and, or, and xor bi-decomposition of  $f$ . For an and bi-decomposition of  $f$ , no variable partition can separate the variable pair  $\{i, j\}$  iff there is at least one or/xor  $2 \times 2$  square in  $f$  associated with  $\{i, j\}$ , i.e., or/xor  $2 \times 2$  squares block the separation of  $i$  and  $j$  for an and bi-decomposition of  $f$ . Similarly, and/xor  $2 \times 2$  squares block the separation of  $i$  and  $j$  in an or bi-decomposition and and/or  $2 \times 2$  squares block the separation of  $i$  and  $j$  in an xor bi-decomposition

of  $f$ . The zero, one, and literal squares are non-blocking for and, or, and xor decompositions. The blocking condition for and, or, and xor bi-decompositions are summarized in Figure 1. The proof for the blocking condition is omitted for brevity. Based on the blocking condition, we will now describe a technique for extracting variable partitions by constructing a BEG for and, or, and xor bi-decompositions.

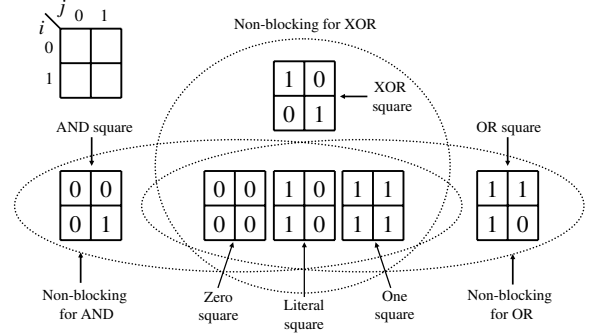


Figure 1: Non-blocking squares for and, or, and xor bi-decompositions

### 2.2 Constructing BEGs

A BEG has one vertex for each input variable of  $f$ . Hence, we will use  $V$  to denote both the input variable set of  $f$  and the vertex set of its BEG. In the BEG of an and decomposition, an edge is inserted between vertices  $i$  and  $j$  if the blocking condition for an and decomposition holds for the variable pair  $\{i, j\}$ . Similarly, an edge is inserted in the BEG of an or (xor) decomposition if the blocking condition for an or (xor) decomposition is satisfied for the variable pair  $\{i, j\}$ . Thus, an edge  $\{i, j\}$  in a BEG means that no variable partition can separate variable  $i$  and variable  $j$ .

For a logic function  $f$  with  $n$  variables, there are  $\binom{n}{2} 2^{n-2} 2 \times 2$  squares, i.e.,  $2^{n-2} 2 \times 2$  squares for each of the  $\binom{n}{2}$  variable pairs. We have developed an efficient algorithm for analyzing the types of these  $\binom{n}{2} 2^{n-2} 2 \times 2$  squares to enable fast construction of the BEGs for and, or, and xor bi-decompositions. Denote the off-set and on-set of a logic function  $f$  by  $f^0$  and  $f^1$ , respectively. Let the function  $x_{\{i,j\}}$  of  $n-2$  variables in  $V \setminus \{i, j\}$  represent all  $2 \times 2$  xor squares associated with  $\{i, j\}$ , i.e., each minterm in  $x_{\{i,j\}}$  is a  $2 \times 2$  xor square associated with  $\{i, j\}$ . Similarly, let  $a_{\{i,j\}}$  and  $o_{\{i,j\}}$  represent the and and or squares associated with  $\{i, j\}$ , respectively. The function  $x_{\{i,j\}}$  can be computed as follows:

$$x_{\{i,j\}} = \forall_i \forall_j (y^1 \cdot z^1) \quad (1)$$

$$\text{where } y^1 = f_i^0 \cdot f_i^1 + f_i^1 \cdot f_i^0 \text{ and } z^1 = f_j^0 \cdot f_j^1 + f_j^1 \cdot f_j^0$$

If  $x_{\{i,j\}}$  is not zero, then there are xor squares associated with  $\{i, j\}$  and hence, edge  $\{i, j\}$  is added in the BEG for the and and or bi-decompositions. Next, the functions  $a_{\{i,j\}}$  and  $o_{\{i,j\}}$  can be computed using functions  $y^1$  and  $z^1$  from Eqn. 1 as follows:

$$a_{\{i,j\}} = (\exists_i \exists_j (f^1 \cdot y^1 \cdot z^1)) \cdot (u \cdot v) \quad (2)$$

$$o_{\{i,j\}} = (\exists_i \exists_j (f^0 \cdot y^1 \cdot z^1)) \cdot (u \cdot v)$$

$$\text{where } y^0 = f_i^0 \cdot f_i^0 + f_i^1 \cdot f_i^1, z^0 = f_j^0 \cdot f_j^0 + f_j^1 \cdot f_j^1,$$

$$u = y_j^0 \cdot y_j^1 + y_j^1 \cdot y_j^0, \text{ and } v = z_i^0 \cdot z_i^1 + z_i^1 \cdot z_i^0$$

If  $a_{\{i,j\}}$  is not zero, then there are and squares associated with  $\{i, j\}$  and hence, edge  $\{i, j\}$  is added in the BEG for the or and xor bi-decompositions. Similarly, if  $o_{\{i,j\}}$  is not zero, then there

are `or` squares associated with  $\{i, j\}$  and hence, edge  $\{i, j\}$  is added in the BEG for the `and` and `xor` bi-decompositions. The BEG for `and`, `or`, and `xor` bi-decompositions is constructed by computing  $x_{\{i,j\}}$ ,  $a_{\{i,j\}}$ , and  $o_{\{i,j\}}$  using Equations 1 and 2 for every pair of variables  $\{i, j\}$ . The largest function we have considered has 149 variables and the CPU time required for constructing the BEG for the `and`, `or`, and `xor` bi-decompositions of this function is 218 secs.

## 2.3 Variable partition

In this section, we will show that the variable partitions of a logic function for the `and`, `or`, and `xor` bi-decompositions can be obtained by analyzing the connectivity of the BEGs. First, we provide a necessary and sufficient condition for the bi-decomposability of a function.

**Theorem 1:** *A logic function  $f$  is not bi-decomposable iff the BEG for the `and`, `or`, and `xor` bi-decompositions are complete graphs.*

**Proof:** The proof is omitted here for brevity.

Theorem 1 states that the bi-decomposability of a function  $f$  can be easily determined using BEGs. In the rest of this section, we describe how variable partitions can be obtained for bi-decomposable functions. We will describe our solution for decomposing functions that are not bi-decomposable in Section 3.2. However, we first describe two commonly used metrics used to measure the quality of a variable partition,  $V_1$  and  $V_2$ , of a logic function  $f$  with a variable set  $V$ ,  $|V| = n$ .

- Total variable count ( $\Sigma$ ): The total variable count,  $|V_1| + |V_2|$ , can range from  $n$  (for a disjoint decomposition) to  $2n - 2$  (for an overlapping decomposition with  $n - 2$  common variables and one unique variable per partition). Variable partitions with lower  $\Sigma$  are preferred since they typically result in decompositions with a small area and power footprint.
- Maximum partition size ( $\Delta$ ): The maximum partition size,  $\max(|V_1|, |V_2|)$ , can range from  $\lceil n/2 \rceil$  (for a balanced disjoint decomposition) to  $n - 1$  (since a bi-decomposition must produce functions that depend on less than  $n$  variables). Variable partitions with lower  $\Delta$  are preferred since they typically result in decompositions with low delay.

We will use  $\mu = [\Sigma, \Delta]$  to measure the quality of a variable partition. Measure  $\mu_1$  is less than measure  $\mu_2$  if either  $\mu_1(\Sigma) < \mu_2(\Sigma)$  and  $\mu_1(\Delta) \leq \mu_2(\Delta)$  or  $\mu_1(\Sigma) \leq \mu_2(\Sigma)$  and  $\mu_1(\Delta) < \mu_2(\Delta)$ . Thus,  $[2n - 2, n - 1]$  is the largest measure for a variable partition.

**Theorem 2:** *A bi-decomposable function  $f$  with variable set  $V$  has an `and` bi-decomposition with the overlapping variable partition  $V_1 = V \setminus \{i\}$  and  $V_2 = V \setminus \{j\}$  iff the edge  $\{i, j\}$  is not present in the BEG for the `and` bi-decomposition.*

**Proof:** The proof is omitted here for brevity.

Theorem 2 also holds for the `or` and `xor` bi-decompositions of  $f$ . Theorem 2 guarantees the existence of an overlapping variable partition for a bi-decomposable function and also shows how the overlapping variable partition can be obtained from the BEG of  $f$ . However, this variable partition may not be the best variable partition for  $f$  since it has the largest possible measure ( $[2n - 2, n - 1]$ ). Before we describe a technique for extracting better variable partitions from the BEGs of  $f$ , we review the definition of a vertex cut in graphs. A vertex cut of a connected graph is a set of vertices whose removal renders the graph disconnected. If  $C$  is a vertex cut of a graph with  $n$  vertices, then any super-set of  $C$  is also

a vertex cut. The maximum size of a vertex cut is  $n - 2$ . Note that a complete graph with  $n$  vertices has no vertex cuts. A minimum vertex cut of a graph is the vertex cut with the smallest size. Note that a graph can have more than one minimum vertex cut. In this paper, the vertex cut for a disconnected graph is assumed to be the empty set ( $\phi$ ).

**Theorem 3:** *If a bi-decomposable function  $f$  has an `and` bi-decomposition with the variable partition  $V_1$  and  $V_2$ , then  $V_1 \cap V_2$  is a vertex cut that disconnects the vertices in  $V_1 \setminus V_2$  from the vertices in  $V_2 \setminus V_1$  of the BEG for the `and` bi-decomposition.*

**Proof:** The proof is omitted here for brevity.

Theorem 3 also holds for the `or` and `xor` bi-decompositions of  $f$ . Using Theorem 3, the vertex cuts of the BEG can be used to obtain variable partitions for the `and`, `or`, and `xor` bi-decompositions of  $f$ . The minimum vertex cuts of the BEG can be used to obtain variable partitions with the smallest  $\Sigma$ . However, the variable partition obtained from minimum vertex cuts may have a large  $\Delta$  since the minimum vertex cut may disconnect the graph into components with unbalanced vertex set sizes. To reduce the value of  $\Delta$ , larger vertex cuts can be chosen (higher  $\Sigma$ ) that disconnect the graph into components with more balanced vertex set sizes.

Our solution to extract variable partitions for a function  $f$  starts with a list of minimum vertex cuts of the BEG for `and`, `or`, and `xor` bi-decompositions. The minimum vertex cut disconnects the BEG into smaller connected components. Larger vertex cuts are obtained by recursively augmenting the vertex cuts with the minimum vertex cut of the largest connected component. The vertex cut with the minimum value of  $\lambda\Sigma + \Delta$ , where  $\lambda$  is a parameter used that determines the relative importance of  $\Sigma$  and  $\Delta$ , is then chosen as the variable partition for  $f$ . The computational details of extracting the minimum vertex cut from an undirected graph are described in Section 4.

## 3. Function decomposition

In the previous section, we have described a technique based on BEGs for extracting variable partitions for `and`, `or`, and `xor` bi-decompositions of a bi-decomposable logic function  $f$ . The first part of this section describes the decomposition of a bi-decomposable function using a determined variable partition,  $V_1$  and  $V_2$ . The second part of this section describes the decomposition of functions that are not bi-decomposable.

### 3.1 Bi-decomposable functions

Denote the off-set and on-set of  $f$  by  $f^0$  and  $f^1$ , respectively, and the off-set and on-set of the decomposed functions for the variable partition  $V_1(V_2)$  by  $f_1^0(f_2^0)$  and  $f_1^1(f_2^1)$ , respectively. Given  $f^0$ ,  $f^1$ , and the variable partition,  $V_1$  and  $V_2$ , we will now describe how  $f_1^0$ ,  $f_1^1$ ,  $f_2^0$ , and  $f_2^1$  can be determined for `and`, `or`, and `xor` bi-decompositions.

**and/or bi-decompositions:** For an `or` bi-decomposition, the on-sets of the decomposed functions,  $f_1^1$  and  $f_2^1$ , are a subset of the on-set,  $f^1$ , of  $f$ . Hence,  $f_1^1$  and  $f_2^1$  can be obtained by expanding the off-set,  $f^0$ , of  $f$  using the existential operator over the variables in  $V \setminus V_1$  and  $V \setminus V_2$  as follows:

$$\begin{aligned} f_1^0 &= \exists_{V \setminus V_1} f^0 \\ f_2^0 &= \exists_{V \setminus V_2} f^0 \end{aligned} \quad (3)$$

Note that the off-sets of the decomposed functions may overlap with the on-set,  $f^1$ , of  $f$ . The on-set,  $f_1^1$  and  $f_2^1$ , for the decomposed functions are obtained by expanding the portion of  $f^1$  that

does not overlap with the off-set  $f_1^0$  or  $f_2^0$  of the decomposed functions, using the existential operator over the variables in  $V \setminus V_1$  and  $V \setminus V_2$ .

$$\begin{aligned} f_1^1 &= \exists_{V \setminus V_1} (f^1 \cdot \overline{f^0}) \\ f_2^1 &= \exists_{V \setminus V_2} (f^1 \cdot \overline{f^0}) \end{aligned} \quad (4)$$

An and bi-decomposition can be obtained in a similar manner by interchanging the off-set and the on-set of  $f$ .

**xor bi-decomposition:** An xor bi-decomposition requires more effort than an and/or bi-decomposition. To obtain an xor bi-decomposition of  $f$  for variable partitions  $V_1$  and  $V_2$ , we use an approach previously proposed in [7] to progressively grow the on-set and off-set of the decomposed functions by adding minterms to cover disjoint portions of the on-set of  $f$ . The pseudocode for the xor bi-decomposition is described in Algorithm 1.

---

#### Algorithm 1: xor bi-decomposition

---

```

input   :  $f^0$  ( $f^1$ ) is the off-set (on-set) of  $f$ 
input   :  $V, V_1, V_2$  are the variable set and the two variable partitions of  $f$ 
output  :  $f_1^0$  ( $f_1^1$ ) is the off-set (on-set) for  $V_1$ 
output  :  $f_2^0$  ( $f_2^1$ ) is the off-set (on-set) for  $V_2$ 

 $f_1^0 = 0, f_1^1 = 0, f_2^0 = 0, f_2^1 = 0$ 
 $g_1^0 = 0, g_1^1 = 0, g_2^0 = 0, g_2^1 = 0$ 
while ( $f \neq 0$ ) do
   $g_1^1 = \text{PickOneCube}(f^1)$ 
  while ( $g_1^0 + g_1^1 \neq 0$ ) do
     $g_2^0 = \exists_{V \setminus V_2} (f^1 \cdot g_1^1 + f^0 \cdot g_1^0)$ 
     $g_2^1 = \exists_{V \setminus V_2} (f^1 \cdot g_1^1 + f^0 \cdot g_1^1)$ 
    if ( $g_2^0 \cdot g_2^1 \neq 0$ ) then
       $\lfloor$  return  $f_1^0, f_1^1, f_2^0, f_2^1 = 0$  /* Variable partition infeasible */
     $f^0 = f^0 - (g_2^0 + g_2^1); f^1 = f^1 - (g_1^0 + g_1^1)$ 
     $f_1^0 = f_1^0 + g_1^0; f_1^1 = f_1^1 + g_1^1$ 
     $g_1^0 = \exists_{V \setminus V_1} (f^1 \cdot g_2^1 + f^0 \cdot g_2^0)$ 
     $g_1^1 = \exists_{V \setminus V_1} (f^1 \cdot g_2^1 + f^0 \cdot g_2^1)$ 
    if ( $g_1^0 \cdot g_1^1 \neq 0$ ) then
       $\lfloor$  return  $f_1^0, f_1^1, f_2^0, f_2^1 = 0$  /* Variable partition infeasible */
     $f^0 = f^0 - (g_2^0 + g_2^1); f^1 = f^1 - (g_2^0 + g_2^1)$ 
     $f_2^0 = f_2^0 + g_2^0; f_2^1 = f_2^1 + g_2^1$ 
  if ( $f^0 \neq 0$ ) then
     $f_1^0 = f_1^0 + \exists_{V \setminus V_1} f^0$ 
     $f_2^0 = f_2^0 + \exists_{V \setminus V_2} f^0$ 

```

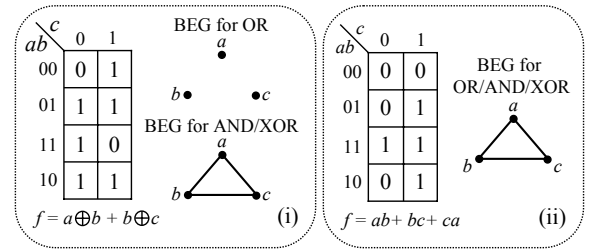
---

**Infeasible variable partitions:** There are some functions for which the variable partitions obtained from the vertex cuts of the BEG are infeasible. A variable partition,  $V_1$  and  $V_2$ , for a function  $f$  is *infeasible* if  $f$  cannot be decomposed into smaller functions with variable sets  $V_1$  and  $V_2$ . The infeasibility of a variable partition of  $f$  can be detected during the decomposition of  $f$  into smaller sub-functions. For an or/and bi-decomposition, if the on-set (off-set) of at least one variable partition is empty and the decomposed functions do not cover the entire on-set (off-set) of  $f$ , then the variable partition is infeasible. For an xor bi-decomposition, if the on-set and off-set of the decomposed functions overlap at any point during the decomposition, then the variable partition is infeasible (see Algorithm 1).

Variable partitions obtained using vertex cuts from a BEG may be infeasible because Theorem 3 only mandates that a vertex cut of the BEG is a necessary, but not sufficient condition for a variable partition of  $f$ . For instance, in Figure 2(a), although the BEG for an or bi-decomposition for  $f$  indicates that a disjoint bi-decomposition exists,  $f$  only has an overlapping or bi-decomposition.

There are two characteristics of  $f$  that together cause the BEG for  $f$  to yield an infeasible variable partition: First, since  $f$  is a symmetric function,  $(\{a, b\}, \{b, c\})$ ,  $(\{a, b\}, \{a, c\})$ , and  $(\{a, c\}, \{b, c\})$  are feasible overlapping variable partitions for an or bi-decomposition of  $f$ . Since there is a variable partition that separates every variable pair, the blocking condition is not satisfied for any variable pair. Hence, there are no edges in the BEG for an or bi-decomposition of  $f$ . Second, since  $f$  is a sparse function (zeros in  $f$  are separated by a Hamming distance of 3) and the edges in the BEG are derived by analyzing  $2 \times 2$  squares (Hamming distance of 2), the symmetry of  $f$  does not manifest itself in the BEG, and thus, the variable partition of  $f$  derived using BEGs is infeasible.

In practice, for various benchmark circuits, we have observed that infeasible variable partitions are rare ( $< 5\%$  cases). Our technique handles an infeasible variable partition for a function by creating an overlapping variable partition,  $V \setminus \{i\}$  and  $V \setminus \{j\}$ , such that  $\{i, j\}$  is not an edge in the BEG. Note that such a  $\{i, j\}$  always exists since the BEG for  $f$  is not a complete graph and Theorem 2 guarantees the validity of the overlapping partition.



**Figure 2: (a) Incorrect disjoint decomposition indicated by BEG and (b) function with a complete BEG.**

### 3.2 Non bi-decomposable functions

Recall that a function is not bi-decomposable if it cannot be decomposed into two functions that each depend on less than  $n$  variables. The BEG for the and, or, and xor bi-decompositions of these functions are complete graphs, and hence there are no vertex cuts for the BEGs. Figure 2(b) illustrates an example of a 3-input function that is not bi-decomposable. Our technique decomposes these functions using an or decomposition. The first function of the or decomposition is obtained by relaxing  $f$  by introducing don't cares. Don't cares are introduced using a universal quantification of  $f$  with a variable  $i$  such that  $\forall_i f$  covers the minimum number of minterms in the on-set. Thus,  $\forall_i f$  is the don't care space for the relaxation of  $f$ . After decomposing the relaxation of  $f$ , the second function of the decomposition is setup to cover the portion of the on-set that was not covered by the first function.

## 4. Results

We will start by illustrating our BEG-based bi-decomposition technique on the 4-input logic function shown in Figure 3(i). First, we construct the BEGs of  $f$  for and, or, and xor bi-decompositions. Since the BEG for and and or bi-decompositions are complete graphs, there are no and or or bi-decompositions for  $f$ . The BEG for the xor bi-decomposition of  $f$  is not a complete graph and has the set  $c$  as a vertex cut of the BEG. Hence, there is an overlapping variable partition  $(\{a, c\}, \{b, c, d\})$  for an xor bi-decomposition of  $f$ , i.e.,  $f = g(a, c) \oplus h(b, c, d)$ . Using algorithm 1, it is determined that the variable partition is feasible and the decomposed functions  $g$  and  $h$  are also obtained. Since  $g$  is the simple two input function  $\overline{a}c$ , Figure 3 does not show the steps for the obtaining the decomposition of  $g$ .

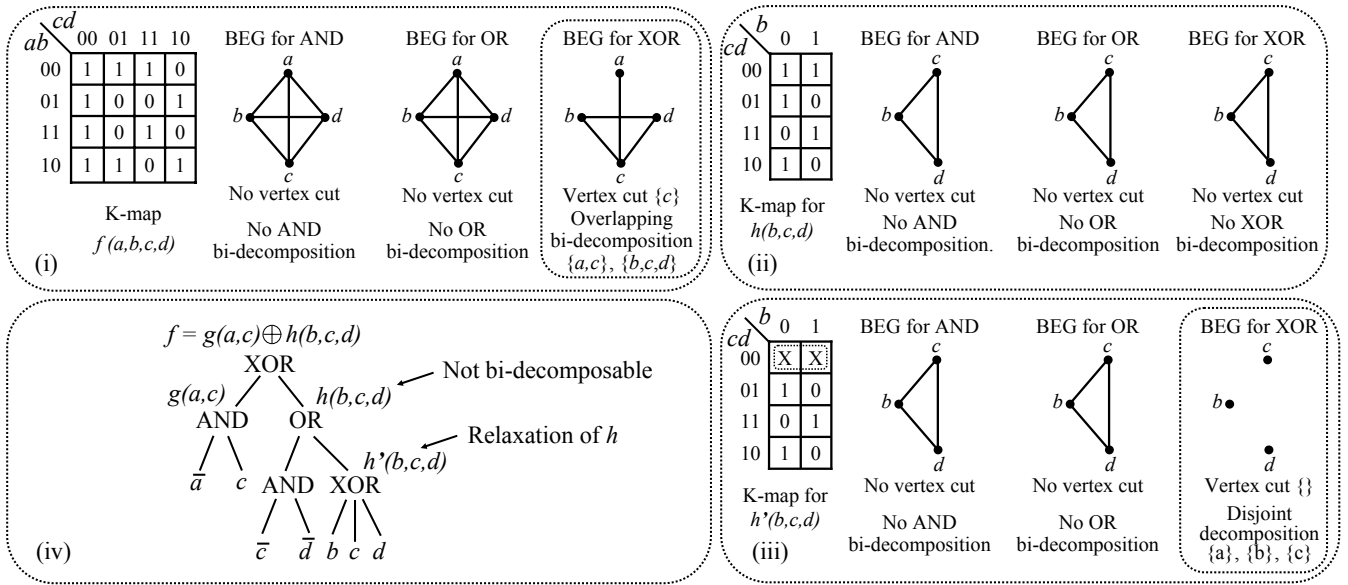


Figure 3: Bi-decomposition using BEGs of (i)  $f(a, b, c, d)$ , (ii)  $h(b, c, d)$ , and (iii)  $h'(b, c, d)$ . (iv) summarizes the bi-decomposition.

The bi-decomposition of  $h$  is the next recursive step and is shown in Figure 3(ii). Since the BEG for the `and`, `or`, and `xor` bi-decompositions of  $h$  are complete,  $h$  is not bi-decomposable. Thus,  $h$  is relaxed to  $h'$  by minimally introducing don't cares in the on-set of  $h$  using a universal quantification of  $h$ ,  $\forall_x h$ , with respect to one variable  $x$ . Since  $\forall_b h$ ,  $\forall_c h$ , and  $\forall_d h$ , cover the same number of minterms in the on-set of  $h$ , we choose  $\forall_b h = \bar{c}\bar{d}$  as the don't care set of  $h'$ . The bi-decomposition for the relaxed function,  $h'$ , has a disjoint `xor` bi-decomposition (see Figure 3(iii)). Thus,  $f = \bar{a}c \oplus (\bar{c}\bar{d} + (b \oplus c \oplus d))$  (see Figure 3(iv)).

## 4.1 Bi-decomposition

Our bi-decomposition technique is implemented within ABC [9]. Given a circuit, each output is represented by BDDs using the CUDD package [10]. Then, each output is recursively decomposed into smaller sub-functions using BEGs. BEGs are stored and manipulated using the `igraph` library [11]. The variable partition for the bi-decomposition of a logic function is obtained from the minimum vertex cuts of the BEG. Our implementation obtains the minimum vertex cut of the BEG of an undirected graph with  $n$  vertices by converting the undirected graph into a directed flow graph with  $2n$  vertices. The minimum edge cut of the directed flow graph, obtained using the algorithm described in [12], is then used to obtain the minimum vertex cut of the undirected graph. The largest function we have considered has 149 variables and the time required to obtain the minimum vertex cut for its BEG is 67 secs.

**Redundancy removal:** Our implementation also performs area recovery using a function-based redundancy removal technique. Since bi-decomposition is performed in a depth-first recursive manner, bi-decomposed functions are cached in a hash table. If the function is encountered again in the same circuit, then the cached bi-decomposition is reused.

We compare our BEG-based bi-decomposition technique to state-of-the-art academic tools — FBDD [4], SIS [8], and ABC [9] — and an industry standard synthesizer. Sixteen circuits from the MCNC, ISCAS, and IWLS benchmark suites and the OpenSPARC T1 processor are optimized using these synthesis tools on a 64-bit 2.4 GHz Opteron-based system. Each benchmark circuit is optimized with each tool to minimize the delay of the decomposed

circuit. The decomposed circuit is mapped to the `lsi_10k` gate library that consists of 89 gates with the industry-standard tool.

The first column in table 1 is the name of the circuit. Subsequent columns report the number of levels of logic in the `and-invert` graph (AIG [9]), the mapped delay, and the dynamic power consumption at 1GHz for the results obtained with each optimization tool. For each benchmark circuit, the *best* results with the lowest mapped delay are reported in the table. For the BDD-based decomposition tool (FBDD), default synthesis options were used. Within SIS, the scripts `delay`, `rugged`, `algebraic`, and `speed_up` were used. Within ABC, script `resyn2rs` was used. Within the industry-standard synthesizer, each design was compiled with the options `-map-effort high` and `-area-effort high` and the design constraint `max_delay` was set to 0. The last row in the table compares the average results across the tools, normalized to the results of the industry-standard tool. On average, our technique shows a 60%, 34%, 45% and 30% reduction in the number of logic levels in the optimized circuit over FBDD, SIS, ABC, and the industry-standard synthesizer, respectively. When mapped delays are evaluated, our technique achieves an average reduction of 20%, 19%, 16% and 20% over the best results of FBDD, SIS, ABC, and the industry-standard synthesizer, respectively. For our technique, the trade-off for a 20% improvement in mapped delay over the industry-standard synthesizer is a 28% increase in the dynamic power consumption.

Table 2 presents results to compare the number of gates in the AIG and the mapped area of our technique with state-of-the-art logic optimization tools. The first two columns in table 2 give the circuit information. Subsequent columns report the number of gates and the mapped area of the logic circuit for each tool.

**Area-delay trade-off:** As discussed in Section 2.3, variable partitions with smaller  $\Sigma$  typically yield bi-decompositions with lower area and power, whereas variable partitions with smaller  $\Delta$  typically yield circuits with lower delay. We have observed that for most circuits the best delay, area, and power is achieved by selecting the variable partition with the smallest  $\Sigma$ . However, some circuits, e.g., `dal`, `sasc`, and `alu2`, exhibit an area versus delay trade-off where reductions in the delay of the decomposed circuit can be achieved when variable partitions with lower  $\Delta$  are chosen.

**Table 1: Comparison of the proposed technique with the best algorithms in FBDD, SIS, ABC, and the industrial tool<sup>†</sup>**

Name	FBDD [4]			SIS [8]			ABC [9]			Industry tool			BEG		
	Levels	Delay	Power	Levels	Delay	Power	Levels	Delay	Power	Levels	Delay	Power	Levels	Delay	Power
cordic	12	3.69	1.47	9	3.64	1.69	10	3.23	2.39	9	3.34	2.44	7	3.35	1.64
dalu	48	7.23	17.8	20	7.17	22.4	31	6.09	25.5	14	7.00	13.9	10	5.49	33.3
t481	6	3.48	0.96	13	3.67	4.2	14	5.59	13.9	11	4.32	7.7	6	3.48	0.96
C432	33	9.63	11.9	22	8.62	15.9	23	9.77	7.9	20	9.84	9.7	11	5.23	36.3
alu2	43	9.07	14.6	22	8.64	13.1	32	8.59	15.7	21	8.69	13.3	10	4.18	13.9
alu4	21	6.25	158.4	12	6.03	55.7	12	6.24	55.9	13	6.11	56.9	10	5.03	87.1
apex4	24	6.2	134.6	12	6.06	55.5	12	6.37	49	13	5.98	60.4	10	5.03	87
term1	18	4.08	6.1	11	3.55	8.2	12	3.69	5.8	10	3.96	7.1	9	3.72	3.2
frg1	13	2.95	1.6	11	3.56	4.1	12	3.71	1.8	10	3.41	4.2	8	2.68	1.7
i7	6	2.23	20.8	5	2.23	42.2	5	2.24	18.3	6	2.23	23.1	5	2.07	20.5
i8	17	5.66	22.7	11	5.51	28.6	11	5.23	39.9	10	5.91	22.6	9	4.7	37.8
too_large	52	6.71	34.1	12	5.01	15.3	24	4.95	13.7	13	5.44	13.3	10	3.91	7.1
lsu_stb_rwctl	17	5.54	31.6	10	5.64	31.6	15	5.68	31.1	11	5.81	29.9	11	5.55	36.5
sparc_tlu_intctl	–	–	–	8	3.44	12.7	11	3.5	11.6	8	3.29	12.1	7	3.01	12.7
sasc	–	–	–	8	3.33	30	8	2.7	28.8	7	3.33	27.3	6	3.11	22.8
spi	–	–	–	–	–	–	28	13.44	78	26	13.06	87.4	18	9.16	193.7
<b>Relative average</b>	2.44	1.25	0.96	1.5	1.23	0.88	1.81	1.18	0.83	1.43	1.25	0.78	1	1	1

<sup>†</sup> FBDD: default; SIS: delay, rugged, algebraic, and speed\_up;

ABC: resyn2rs; Industry-standard synthesizer: -map-effort high -area-effort high and set\_max\_delay 0

**Table 2: Comparison of the proposed technique with the best algorithms in FBDD, SIS, ABC, and the industrial tool**

Name	PI/POs	FBDD [4]		SIS [8]		ABC [9]		Industry tool		BEG	
		Gates <sup>‡</sup>	Area <sup>*</sup>	Gates <sup>‡</sup>	Area <sup>*</sup>	Gates <sup>‡</sup>	Area <sup>*</sup>	Gates <sup>‡</sup>	Area <sup>*</sup>	Gates <sup>‡</sup>	Area <sup>*</sup>
cordic	23/2	62	99	88	98	53	151	65	146	32	105
dalu	75/16	1201	1190	1604	1426	1046	1515	707	1022	862	1911
t481	16/1	25	72	1227	202	742	936	159	510	25	72
C432	36/7	235	492	273	399	136	437	218	443	826	1487
alu2	10/6	484	716	482	614	349	748	357	608	400	778
alu4	9/19	4771	5619	2194	4455	2007	4372	1610	4676	2356	5218
apex4	9/19	4540	5286	2194	4653	2003	3681	1589	4280	2349	5218
term1	34/10	262	338	323	440	147	346	186	443	94	270
frg1	28/3	51	101	110	226	84	176	130	227	51	101
i7	199/67	510	1365	650	1431	568	1234	510	1629	500	1692
i8	133/81	979	1291	1621	1435	892	1635	1212	1180	1101	1786
too_large	38/3	1585	1639	614	894	392	748	434	695	172	434
lsu_stb_rwctl	250/205	490	1029	562	1095	482	998	587	969	755	1272
sparc_tlu_intctl	82/80	–	–	227	739	174	682	241	711	221	794
sasc	250/132	–	–	776	1451	547	1453	683	1374	536	1072
spi	505/227	–	–	–	–	3158	3888	3083	4131	4623	8274
<b>Relative average</b>	–	1.95	1.1	4.82	1.2	2.87	1.74	1.54	1.4	1	1

<sup>‡</sup>The number of gates is reported after decomposition as the number of nodes in the AIG.

<sup>\*</sup>Area is reported after the decomposed AIG is mapped using the industrial tool in all cases.

## 5. Conclusions

This paper described a new approach for obtaining variable partitions for the bi-decomposition of logic functions. Disjoint and overlapping variable partitions for and, or, and xor bi-decompositions of a logic function were obtained from the vertex cuts of an undirected graph called the *blocking edge graph*. Using this technique, an average reduction in delay of 20% was achieved for an average power overhead of 28% over the best results of an industry-standard synthesis tool across 16 benchmark circuits.

## References

- [1] G. de Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.
- [2] G. Hachtel and F. Somenzi, *Logic synthesis and verification*. Kluwer Academic Publishers, 2000.
- [3] C. Yang *et al.*, “BDS: A BDD-based logic optimization system,” *IEEE Trans. Computer-aided Design*, vol. 21, pp. 866–876, 2000.
- [4] D. Wu *et al.*, “FBDD: A folded logic synthesis system,” in *Intl. Conference on ASIC*, pp. 746–751, 2005.
- [5] A. Mishchenko, B. Steinbach, and M. Perkowski, “An algorithm for bi-decomposition of logic functions,” in *Proc. Design Automation Conference*, pp. 103–108, 2001.
- [6] H.-P. Lin, J.-H. R. Jiang, and R.-R. Lee, “To SAT or not to SAT: Ashenurst decomposition in a large scale,” in *Proc. Intl. Conference Computer-aided Design*, pp. 32–37, 2008.
- [7] B. Steinbach, “Synthesis of multi-level circuits using exor-gates,” in *Proc. Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 161–168, 1995.
- [8] E. Sentovich *et al.*, “SIS: A system for sequential circuit synthesis,” Tech. Rep. UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.
- [9] “ABC Logic synthesis tool.” Please visit the URL <http://www.eecs.berkeley.edu/~alanmi/abc/> for further details.
- [10] CUDD: Colorado University Decision Diagram Package. Please visit the URL <http://vlsi.colorado.edu/~fabio/CUDD/> for further details.
- [11] The igraph library for complex network research. Please visit the URL <http://igraph.sourceforge.net/> for further details.
- [12] Y. Boykov *et al.*, “An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 26, pp. 1124–1137, 2004.