Timing-driven N-way Decomposition

David Baneres
Univ. Oberta de Catalunya
Barcelona, Spain

Jordi Cortadella
Univ. Politècnica de Catalunya
Barcelona, Spain

Mike Kishinevsky[*]
Strategic CAD Lab, Intel Corp.
Hillsboro, OR USA

ABSTRACT

Logic decomposition has been extensively used to optimize the worst-case delay and the area in the technology independent phase. Bi-decomposition is one of the state-of-art techniques to reduce the depth of the netlist due to the affordable computational cost. We present a novel n-way decomposition technique that improves bi-decomposition. The problem of decomposition is formulated as a Boolean relation which captures a larger set of possible solutions compared to bi-decomposition. The solution obtained from the Boolean relation improves the delay with near-zero cost in area. As it is shown on the experimental results, a considerable improvement is achieved on large netlists and even larger depending on which technology mapper is used.

Categories and Subject Descriptors: B.6.3 [Hardware]: Logic Design - Design Aids; J.6 [Computer Applications]: Computer-aided engineering

Terms: Algorithms, Design.

Keywords: Logic design, timing optimization, decomposition.

1. INTRODUCTION

Logic decomposition is a logic transformation that has been extensively used in multi-level minimization [1–4]. Mostly, this transformation has been used aiming at reducing the area of Boolean networks, since common factors from different functions can be shared during the decomposition. Moreover, logic decomposition has been also applied targeting to other optimization objectives like timing [5] or layout [6].

Logic decomposition mostly depends on the initial structure of the network. Sharing common factors is more likely in networks with large functions on the nodes. Therefore, some techniques perform a partial collapse of the network, or even a complete collapse of the primary outputs, to obtain more factors. In this paper, we refer to timing-driven recursive decomposition methods applied to entire or partial collapsed networks. The decomposition is recursively applied from the functions of the collapsed nodes until the primary inputs are reached. The main objective is to decrease the depth of the netlist. The depth is not an accurate estimation of the final circuit delay but both have a high correlation.

[*]This work was supported in part by CICYT TIN2007-66523 and by a grant from Intel Corporation.

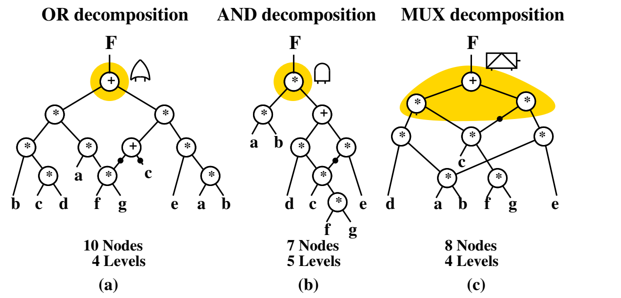

Figure 1: Comparison between (a,b) bi-decomposition and (c) n-way decomposition.

In timing-driven optimization, several methods have been proposed to perform the decomposition. A non-disjoint technique is presented in [2]. This approach explores all possible decompositions with the AND and XOR logic functions. This exact technique obtains high quality results but incurs on large runtimes because of the extensive exploration. BDD decomposition [3, 4] approaches have been also proposed to avoid the wide exploration. They combine BDD manipulation of the functions with the sharing between internal equivalent functions.

The technique proposed in [7] combines the best characteristics of each previous approach: algebraic and BDD decomposition, and function approximation are explored. Sharing is also applied to reduce the total area. Moreover, the solution is balanced by using tree-height reduction techniques [8] after the decomposition.

This paper presents a timing-driven n-way decomposition method. The problem of decomposition is formulated as a Boolean relation which can be solved towards a specific objective. The presented algorithm shows improvements on area and delay with regard the bi-decomposition method presented in [7]. The drawback is that the range of application is limited to small- and medium-size networks due to the complexity of the decomposition problem with Boolean relations.

This paper is organized as follows. An overview of the n-way decomposition approach is presented in Section 2. Section 3 introduces the basic background and terminology that is used in the paper. Section 4 presents the n-way decomposition method and the implementation aspects are described in Section 5. Finally, the experimental results are reported in Section 6.

2. OVERVIEW

In this section, an example of decomposition is presented. Let us assume the next Boolean function:

$$F(a,b,c,d,e,f,g) = abcdfg + ab(\overline{cfg})e$$

Figure 1 illustrates three possible decompositions using the OR2, AND2 and MUX functions. The decompositions are represented in a directed acyclic graph where the nodes represent two-input logic functions and a bubble on an edge corresponds to an inverter. The shadowed nodes represent the functions used to obtain the decomposition (we assume that the MUX is internally represented by two-input gates). The OR2 function requires one less level of logic compared to the AND2 function, but a solution with more nodes is produced. The MUX finds an intermediate solution between them because the n-way method explores a larger search space of possible decompositions compared to the AND2 and OR2 decompositions. The MUX solution can be also retrieved by the OR2 function but it is more difficult to find because sharing between common subexpressions must be detected.

The contributions of this paper are detailed next:

- We generalize the bi-decomposition problem to n-way decomposition.
- The problem is formulated as a Boolean relation [9] which can be solved targeting to a specific objective. In our case, the objective is set to improve the depth of the functions.
- N-way decomposition explores a huge space of potential solutions. The selected solution improves the delay with near-zero overhead in area with regard to bi-decomposition as it is shown in the experimental results.

3. BACKGROUND

In this section, we review the definition of Boolean relation. Next, the n-way decomposition problem is formulated and the representation for a decomposed netlist is described.

3.1 Review of Boolean relations [9]

Let us define a Boolean function and a multiple-output Boolean function.

DEFINITION 3.1. Boolean function. *A Boolean function f is a function $f : \mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B} = \{0,1\}$. A Boolean function can also be interpreted as the set of vertices $x \in \mathbb{B}^n$ such that $f(x) = 1$.* □

DEFINITION 3.2. Multiple-output Boolean function. *A multiple-output Boolean function f is a function $f : \mathbb{B}^n \to \mathbb{B}^m$. It can be also specified as a vector of functions $f = (f_1, f_2, \dots, f_m)$.* □

A Boolean relation is defined as follows:

DEFINITION 3.3. Boolean relation. *A Boolean relation R is a subset of $\mathbb{B}^n \times \mathbb{B}^m$, where $\mathbb{B}^n$ and $\mathbb{B}^m$ are called the input set and output set of R, respectively. It can be specified by a characteristic function $R : \mathbb{B}^m \times \mathbb{B}^n \to \mathbb{B}$, such that $(x,y) \in R$ if and only if $R(x,y) = 1$. A Boolean relation is* well defined *if for all $x \in \mathbb{B}^n$, there is $y \in \mathbb{B}^m$ such that $(x,y) \in R$.* □

A well-defined Boolean relation covers a set of multiple-output Boolean functions.

DEFINITION 3.4. Compatible functions. *Given a relation R, the set of multiple-output functions compatible with R is defined as*

$$\mathbb{F}(R) = \{F \mid F \subseteq R \wedge F \textit{ is a multiple-output function}\}$$

A function F is compatible iff $F \cap \overline{R} = \emptyset$. □

The objective of a Boolean relations solver is to seek for one of the best multiple-output compatible functions with respect a optimization objective (e.g. function with the smallest depth).

3.2 Problem formulation

This section describes the n-way decomposition problem.

DEFINITION 3.5. N-way decomposition. *Let us assume a function $F(X)$ with the set of input variables $X = \{x_1, x_2, \cdots, x_m\}$ and a function $G(Y)$ with the set $Y = \{y_1, y_2, \cdots, y_n\}$. The* n-way decomposition *of the function $F(X)$ with regard $G(Y)$ is a multiple-output function $DEC = (F_1(X), F_2(X), \dots, F_n(X))$ such that $F(X) = G(F_1(X), F_2(X), \dots, F_n(X))$, where $F_1, F_2, \dots, F_n$ are functions with the same set of variables X than the function F.* □

Bi-decomposition is a particular case of n-way decomposition where $G(Y)$ is a two-input function. Hereafter, we will also refer to bi-decomposition and n-way decomposition as BiDec and NDec respectively. The n-way decomposition problem can be naturally represented by a Boolean relation which covers the potential set of decompositions.

DEFINITION 3.6. Decomposition problem formulation. *Let us assume a function $F(X)$ with the set of variables $X = \{x_1, x_2, \cdots, x_m\}$ and a function $G(Y)$ with the set $Y = \{y_1, y_2, \cdots, y_n\}$. The well-defined Boolean relation that represents all possible decompositions of the function $F(X)$ with respect $G(Y)$ is defined as follows:*

$$R(X,Y) = (F(X) \Leftrightarrow G(Y)) + DC(X)$$

where X and Y corresponds to the input and output set of the relation, respectively. The $DC(X)$ is the external don't care set of the function $F(X)$. □

The function being decomposed usually belongs to a network where flexibility described with don't cares can be computed. The don't care information is added to the problem to increase the flexibility of the search with the decompositions covered in other regions of the network.

EXAMPLE 3.1. *The example of a decomposition using a multiplexer shown in Fig. 1(c) is illustrated next. Let us assume the Boolean function $F(a,b,c,d,e,f,g)$ described in the previous section and the function of the multiplexor $MUX(A,B,C) = AB + \overline{A}C$. We also assume that the don't care set of F is empty.*

The characteristic function of the Boolean relation that captures the flexibility of the implementation is

$$\begin{aligned} R(a,b,c,d,e,f,g,A,B,C) &= F(a,b,c,d,e,f,g) \Leftrightarrow MUX(A,B,C) \\ &= (abcdfg + ab(\overline{cfg})e)(AB + \overline{A}C) \\ &+ \overline{(abcdfg + ab(\overline{cfg})e)}(A\overline{B} + \overline{A}\,\overline{C}) \end{aligned}$$

The Boolean relation R covers the set of all possible decompositions using a multiplexer. If the Boolean relation is solved with a Boolean relations solver [9], part of the original function F is absorbed within the multiplexer. Figure 1(c) shows the following decomposition that could be obtained with the solver $MUX_{dec} = (A_{dec}, B_{dec}, C_{dec})$ where

$$\begin{aligned} A_{dec}(a,b,c,d,e,f,g) &= cfg; \\ B_{dec}(a,b,c,d,e,f,g) &= abd; \\ C_{dec}(a,b,c,d,e,f,g) &= abe \end{aligned}$$

The multiple-output function MUX_{dec} is included in the set of compatible functions of the Boolean relation R, because $MUX_{dec} \cap \overline{R} = \emptyset$. □

3.3 Network representation

A decomposed netlist is a graph where each node represents a function. In this paper, a BDAG representation is used.

Algorithm 1 BRCDEC: Algorithm for logic n-way decomposition.

```
Input: Function $F$ to be decomposed, don't cares $DC$,
    $library$ of logic functions to perform the n-way decomposition,
    required time $RT$.
Output: Function recursively decomposed in BDAG representation.
1:  Collapse($F$)
    {Obtain bi-decomposition}
2:  Decomposition: $dec \Leftarrow$ Obtain bi-decomposition($F$, $DC$, $RT$)
3:  List of decompositions: $List \Leftarrow Tree\text{-}height\ reduction(dec)$
    {Obtain decompositions using the logic functions of $library$}
4:  for each function $G$ in $library$ do
5:      $dec \Leftarrow$ Obtain n-way decomposition($F$, $G$, $DC$, $RT$)
6:      $List \Leftarrow List \cup Tree\text{-}height\ reduction(dec)$
7:  end for
    {All decompositions are in BDAG representation}
8:  Best Decomposition: $Sdec \Leftarrow$ Select Best Decomposition($List$)
9:  if Levels($Sdec.left$)>Levels($Sdec.right$) then
10:     swap($Sdec.left$, $Sdec.right$)
11: end if
    {First descompose the child with less number of levels}
12: Decomposition: $Ldec \Leftarrow$ BRCDEC($Sdec.left$, $DC$, $library$, $RT$-1)
    {Add observability don't cares}
13: Decomposition: $Rdec \Leftarrow$ BRCDEC($Sdec.right$, $DC$+$ODC(Ldec)$, $library$, $RT$-1)
14: return Create decomposition of function $F \Leftarrow$ ($Sdec.op$, $Ldec$, $Rdec$)
```

DEFINITION 3.7. Binary Directed Acyclic Graph (BDAG). *A Binary DAG is a directed acyclic graph, in which a node has either 0 or 2 incoming edges. A node with no incoming is a primary input. A node with 2 incoming edges is a two-input AND or OR node. An edge is either complemented or not. A complemented edge, represented with a bubble, indicates the inversion of the signal.* □

Each node of the BDAG represents a single bi-decomposition.

DEFINITION 3.8. Bi-decomposition representation. *A bi-decomposition D can be represented as a triples*

$$D \equiv (op, left, right)$$

where op is the Boolean function $G(y_1, y_2)$ used in the decomposition, and $left$ and right are the resulting functions for y_1 and y_2. op is the Boolean operator of the node that can be either $*$, $+$ or input if the node is a literal. Recursively, $left$ and right are also defined as other triples. □

The decompositions shown in Fig. 1(c) are represented as BDAGs. Note that, an n-way decomposition can be also represented as a BDAG if the function used to perform the decomposition is also represented in two-input nodes.

In a BDAG, the delay of a function can be easily estimated. Each node g has a delay or *arrival time* ($AT(g)$) based on the worst-case delay needed to obtain the correct value. The *required time* ($RT(g)$) is defined as the time when the correct value is expected at the output of the node. In the technology independent phase, we define the arrival time as the maximum depth from the node g to the primary inputs. The required time specifies the maximum depth that should have the node to meet the cycle time.

4. RECURSIVE N-WAY DECOMPOSITION

In this section the recursive n-way decomposition procedure is presented. Algorithm 1 illustrates the method called BRCDEC (*Boolean Relation Combinational DEComposition*). The input of the procedure is the function F to be decomposed and the don't care information DC captured from the environment. A library is also provided with the set of logic functions to perform the NDec. The last input specifies the desired required time.

Initially, the function is collapsed (Line 1) and the bi-decompositions obtained by [7] are explored. This includes algebraic and BDD decomposition based on function approximation (Line 2).

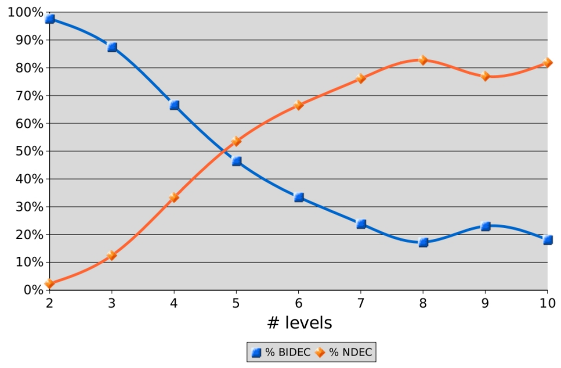

Figure 2: Percentage of selection of the type of decomposition by number of levels of the decomposed function.

The next step is to explore the decompositions produced using the set of logic functions of the library (Lines 4-7). The decomposition problem is formulated as a Boolean relation as we defined in Def. 3.6 and the relation is solved with a BR solver[1]. Note that a Boolean relation covers a huge space of decompositions and the calculation of the best solution may incur on large runtime penalties. In order to cut the runtime, the solver is limited to explore a small set of decompositions.

Our approach improves the technique described in [7] by providing better decompositions using Boolean relations. In the experimental results, the logic functions `AND2`, `OR2`, `MUX`, `XOR`, `OA22` ($x_1x_2 + x_3x_4$), `AO22` ($(x_1 + x_2)(x_3 + x_4)$) have been selected for the library. Notice that the library also includes the AND2 and OR2 function processed by BiDec. The formulation of the BiDec problem as a Boolean relation is able to produce sometimes a better decomposition, mostly in large functions.

Experimentally, we have observed there is a relationship between the selected decomposition (BiDec or NDec) and the number of levels of the logic function. Figure 2 shows a plot between the number of levels of the function before decomposition and the percentage of selection of the type of decomposition. The plot is obtained by decomposing around 500 functions of several levels. Note that, BRCDEC is not recursively called on this experiment. The objective is to show the percentage of selection on individual decompositions. When the function has few levels, there are few feasible decompositions, therefore, a near-optimal solution is easily obtained by using BiDec. NDec obtains decompositions with a smaller number of levels compared to BiDec when the complexity of the function grows. The accuracy of bi-decomposition is dramatically affected by the complexity of the function. Decompositions provided by the n-way method are mostly selected on function with a large number of levels, as it is shown in the plot, since a larger set of candidates is explored.

Tree-height reduction [7, 8, 10] is applied as a post-process in both methods (BiDec and NDec) before the selection of the decomposition with the smallest number of levels. The objective is twofold. First, the decomposition is refined. The solution is balanced improving the number of levels. Moreover, the decomposition is represented as a BDAG where the area and the delay can be easily estimated.

Next, the algorithm selects the best decomposition (Line 8). The selection is performed evaluating their BDAG representation using the cost function defined in Section 5.2.

Finally, the algorithm recursively decomposes the `left` and

[1]The configuration of the solver is described in the following section.

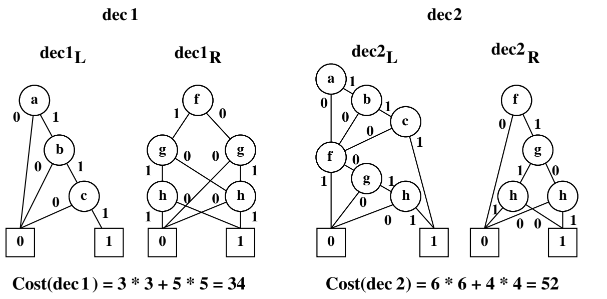
dec 1
dec2
dec1L
dec1R
dec2L
dec2R
Cost(dec 1) = 3 * 3 + 5 * 5 = 34
Cost(dec 2) = 6 * 6 + 4 * 4 = 52

Figure 3: Sum of the squares of the BDD sizes cost function.

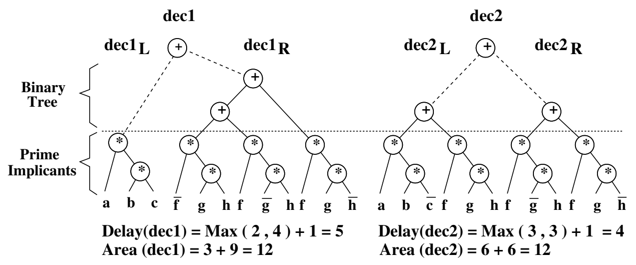

Figure 4: Cost function based on prime implicants.

`right` function (Lines 12-13). The observability don't cares from one function are applied to the other one to avoid the redundancy between them. First, the smaller one is processed. This order contributes to reduce the size of the larger function. When `left` and `right` are completely decomposed, the decomposition for the target function F is constructed (Line 14).

The algorithm describes the NDec process on a single function. A netlist is composed of a set of primary outputs with their respective Boolean functions. To recursively decompose the netlist, the algorithm is applied to the functions of all the primary outputs. The technique also detects isomorphic functions to avoid multiple decompositions of the same function.

5. IMPLEMENTATION ASPECTS

This section describes the configuration of the solver and the cost functions.

5.1 BREL solver

The BREL solver proposed in [9] is used to solve the Boolean relations. The solver has been customized to support two cost functions. The first function filters the huge space of solutions using its BDD representation and a small set of solutions are selected. The latter one selects the final decomposition using the BDAG representation over the best solutions found by the former cost function. The solver has been limited to perform a partial exploration of 200 decompositions for each Boolean relation due to the large space of potential solutions.

5.2 BREL cost functions

The former cost function is based on BDD representation to obtain a fast estimation. BDD-based cost functions are not accurate, since sometimes there is no correspondence between the complexity of the BDD and the function. Another drawback is that the arrival time of the inputs can not be taken into account. The function being decomposed usually belongs to a larger network. The input variables also have timing information which has to be taken into consideration to obtain a good estimation.

The computation of the number of levels of a function is difficult to be estimated in BDD representation. Although a BDD can be transformed to a BDAG to increase the accuracy, it is not recommended since experimentally we observed that the execution of BREL is slowed down considerably.

A naive BDD-based cost function is presented in [9]. The balance of a decomposition is estimated as the sum of squares of the BDD sizes. This approximation gives an intuition of the quality of the decomposition.

EXAMPLE 5.1. *Consider the bi-decomposition using an OR function of $F = abc + \overline{f}gh + f\overline{g}h + fg\overline{h}$. Let us assume the same arrival time for all the inputs. Two possible decompositions are:*

$$dec1 \equiv (+, dec1_L, dec1_R) \quad \Rightarrow \quad \begin{cases} dec1_L = abc, \\ dec1_R = \overline{f}gh + f\overline{g}h + fg\overline{h} \end{cases}$$

$$dec2 \equiv (+, dec2_L, dec2_R) \quad \Rightarrow \quad \begin{cases} dec2_L = abc + \overline{f}gh, \\ dec2_R = f\overline{g}h + fg\overline{h} \end{cases}$$

Figure 3 shows the BDD representation for these two decompositions. The BDD cost function "sum of squares" selects $dec1$ instead of the more balanced decomposition $dec2$. □

The next BDD-based cost function is proposed to obtain a more accurate estimation of the delay and area. A balanced binary tree is constructed from the disjunction of all prime implicants of the function where each prime implicant is a conjunction of input variables. The arrival time of the inputs is used to build the tree towards a balanced delay. The delay and the area are evaluated during the computation of the prime implicants. The delay is estimated by the depth of the tree and the area by the sum of the support of the primes.

DEFINITION 5.1. BDD-based cost function. *The cost is defined as the pair $C = (Delay, Area)$. A decomposition Bdec with the cost C_{Bdec} is the best one if there is no other decomposition Dec with the cost C_{Dec} in the set of explored decomposition such that $Delay_{Dec} < Delay_{Bdec}$ or $Delay_{Dec} = Delay_{Bdec} \wedge Area_{Dec} < Area_{Bdec}$.* □

EXAMPLE 5.2. *Consider the same decompositions of Fig. 3. Figure 4 depicts both decompositions in binary tree representation where the cost function based on prime implicants is used. The selection of the best solution changes since the cost of each decomposition is $C_{dec1} = (5, 12)$ and $C_{dec2} = (4, 12)$.* □

This BDD-based cost function, which is used as the first cost function in BREL, reduces the possible decompositions to a small set of candidates. Another cost function based on BDAG representation is used to perform the final selection over the set of candidates. Notice that the same cost function is used in the algorithm BRCDEC to select the best decomposition in Select Best Decomposition (Line 8).

The complexity is significantly higher but the estimation is more accurate and the computational cost is affordable since a small set of the solutions is processed. The cost is also defined as a pair $C = (D, A)$ where the delay D and area A are calculated as the depth and the number of nodes of the decomposition respectively. The delay and the area are computed as described next:

$$D(dec) = \begin{cases} AT(dec) & \text{if } dec.op = input \\ 1 + max(D(dec.left), D(dec.right)) & \text{otherwise} \end{cases}$$

where $AT(dec)$ is the arrival time when the node is a primary input.

			LEVELS		DELAY			AREA			CPU	
	PI	PO	BiDec	NDec	BiDec	NDec	$-\Delta(\%)$	BiDec	NDec	$-\Delta(\%)$	BiDec	NDec
9symml	9	1	9	9	7.17	6.73	6.14	117392	131776	-12.25	6	196
alu2	10	6	11	9	8.91	8.15	8.53	509008	475136	6.65	13	579
apex6	135	99	7	7	7.39	6.92	6.36	1529808	1517280	0.82	11	788
apex7	49	37	8	7	7.67	7.57	1.30	881136	919648	-4.37	7	511
b9	41	21	6	5	5.29	5.14	2.84	212976	218080	-2.40	1	64
c8	28	18	6	6	5.50	5.55	-0.91	212048	207408	2.19	1	57
cht	47	36	4	4	4.94	4.71	4.66	272368	270048	0.85	1	29
count	35	16	7	7	6.46	6.34	1.86	497408	517360	-4.01	3	142
cu	14	11	5	5	4.70	4.73	-0.64	95120	100688	-5.85	1	14
example2	85	66	6	6	8.54	7.49	12.30	1331680	1288528	3.24	7	379
f51m	8	8	7	7	6.43	6.14	4.51	219008	210192	4.03	2	85
frg1	28	3	7	7	5.40	5.41	-0.19	72384	81200	-12.18	3	121
i5	133	66	6	6	6.28	6.23	0.80	1013376	990176	2.29	6	363
i7	199	67	5	5	6.83	7.10	-3.95	833808	942384	-13.02	4	251
lal	26	19	6	5	5.30	5.19	2.08	220864	230144	-4.20	1	59
pcle	19	9	6	6	5.48	5.36	2.19	139664	150336	-7.64	1	43
pcler8	27	17	5	5	5.77	5.73	0.69	229216	237104	-3.44	1	69
sct	19	15	5	5	4.90	4.77	2.65	167040	161008	3.61	1	43
term1	34	10	9	9	6.89	6.93	-0.58	361456	353568	2.18	4	288
ttt2	24	21	6	6	6.40	6.13	4.22	362848	388368	-7.03	2	139
x1	51	35	6	6	5.76	5.62	2.43	487200	498336	-2.29	4	279
x2	10	7	5	5	4.46	4.57	-2.47	62640	62176	0.74	1	16
x3	135	99	7	7	6.92	6.82	1.45	1543728	1525632	1.17	11	16
x4	94	71	5	5	6.74	6.63	1.63	1046784	1056064	-0.89	6	424
z4ml	7	4	5	5	4.98	4.90	1.61	82128	84912	-3.39	1	27
Norm.			1.000	0.972	1.000	0.978		1.000	1.004			

Table 1: Comparison with the bi-decomposition method presented in [7] on small networks.

$$A(dec) = \begin{cases} 1 & \text{if } dec.op = input \\ A(dec.left) + A(dec.right) & \text{otherwise} \end{cases}$$

The same cost function defined on Def. 5.1 over the cost $C = (D, A)$ is used to select the best decomposition. The cost function is more accurate since they work on top of the BDAG representation of the decomposition. Note that, the detected isomorphic functions are merged in a BDAG that contributes to obtain a better estimation of the area of the function.

6. EXPERIMENTAL RESULTS

Two experiments have been performed to show the efficiency of the n-way decomposition approach presented on this paper:

- Comparison with the bi-decomposition proposed in [7] on small- and medium-size netlists of MCNC benchmarks.
- Comparison with bi-decomposition on industrial circuits.

Our approach is not compared to other timing-driven techniques because an exhaustive comparison has been previously performed in [7]. BRCDEC has been implemented in SIS using the BREL solver [9]. A library of logic functions has been provided to BRCDEC to compute several decompositions. Specifically, the library consists of six functions: AND2, OR2, AO22, OA22, MUX, and XOR. Experimentally, we observed these functions provide the maximum number of distinct decompositions. More functions can be used to increase the solution space. However, the exploration time would be also affected.

6.1 Comparison on MCNC benchmarks

In this section, the comparison with BiDec [7] is performed. The experiment is run on a subset of small and medium-size circuits of the MCNC benchmarks. The objective of this experiment is to show the performance of BRCDEC with regard to bi-decomposition.

The tables of results report for each example the number of primary inputs and outputs, the number of levels of logic in the technology independent phase, the delay and area after technology mapping and the runtime. The circuits have been mapped using the tree-mapping map -AFG [11, 12] targeting at delay optimization with the academic library lib2.genlib. The last row of the tables reports the normalized sum of the columns.

Table 1 summarizes the results on the small netlists. A subset of the netlists reported in [7] is selected. The smallest ones have been removed since the decomposition obtained with BRCDEC is identical to the solution provided by bi-decomposition.

The number of levels of logic is similar in both methods. BRCDEC reduces the depth of the circuits by 3%. However, the number of levels is actually reduced only on few examples (alu2, apex6 and b9). The improvement is closely related to the size of the network. For instance, the number of levels is reduced from 11 to 9 in the circuit alu2. However, BRCDEC contributes to obtain better delay after technology mapping with similar results on area. In some examples the results are substantially better (alu2, apex6, apex7 and example2). Only, the results are slightly worst in four examples (c8, i7, term1 and x2), since a bad decision is taken during the recursive decomposition. Nevertheless, there is an improvement of 2% on average, since BRCDEC generates some decompositions that bi-decomposition is unable to find. However, the runtime of BRCDEC using Boolean relations is considerably higher in comparison to [7].

These statements are confirmed in Table 2. This table reports the results on the largest MCNC netlists that can be run with BRCDEC without incurring on large penalties on runtime. On larger ones, BRCDEC blows up due to the construction of too large BDDs. Here, the improvement in number of levels is similar to the previous table. However, the improvement on delay after technology mapping is more significant (5%) with a slightly reduction on the area. Performing an analysis on individual netlists, we have observed that BRCDEC is able to find decompositions with similar number of levels but with less area on large functions (i8, i9 and vda). On the recursive decomposition, the n-way decomposition is commonly selected on functions with larger depth. Moreover, the obtained decompositions tend to share more common subexpressions that contributes to slightly reduce the area of the circuits.

			LEVELS		DELAY			AREA			CPU	
	PI	PO	BiDec	NDec	BiDec	NDec	$-\Delta(\%)$	BiDec	NDec	$-\Delta(\%)$	BiDec	NDec
frg2	143	139	8	8	9.38	8.99	4.16	4139808	4360672	-5.34	47	4284
i8	133	81	9	8	10.37	9.35	9.84	4084128	3849344	5.75	64	3691
i9	88	63	8	8	10.75	10.18	5.30	4447904	4382016	1.48	54	4146
table3	14	14	10	10	10.67	10.05	5.81	3216912	3258208	-1.28	121	3580
vda	17	39	8	8	8.70	9.01	-3.56	2219776	2093568	5.69	24	1307
Norm.			1.000	0.976	1.000	0.954		1.000	0.991			

Table 2: Comparison with the bi-decomposition method presented in [7] on medium-sized networks.

		LEVELS		CELLS		AREA		WNS		TNS	
		Bidec	Ndec	Bidec	Ndec	Bidec	Ndec	Bidec	Ndec	Bidec	Ndec
100 ps	Netlist1	6	5	153	137	689	602	-12	1	-24	0
	Netlist2	5	5	253	182	1234	855	-17	-11	-80	-20
	Netlist3	5	5	210	124	1073	579	-19	-16	-102	-28
	Netlist4	6	5	363	316	1855	1655	-28	-18	-312	-189
150 ps	Netlist5	5	5	292	251	895	828	9	12	0	0
	Netlist6	5	6	572	472	2168	1754	-4	-8	-13	-25
	Netlist7	6	6	477	463	2284	2168	-22	-16	-117	-82
	Netlist8	6	6	498	468	2342	2136	-20	-17	153	-115
200 ps	Netlist9	7	7	1719	1750	6426	6641	-15	-9	-51	-58
	Netlist10	7	7	2077	1512	8358	5237	-19	-5	-156	-7
	Netlist11	9	7	6001	1613	39023	5990	-194	-8	-6564	-42
	Norm.	1.000	0.982	1.000	0.866	1.000	0.823	1.000	0.597	1.000	0.524

Table 3: Comparison with the bi-decomposition method presented in [7] on industrial circuits.

6.2 Comparison on industrial circuits

In this section, the comparison between both techniques is performed on industrial circuits. We have selected 11 netlists and a library of 90*nm* from Intel Corporation. Instead of using the tree-mapper of SIS, a graph-mapper [13] is applied. This technology mapper takes better advantage of the graph decomposition performed by both techniques.

Table 3 summarizes the results. The table reports the number of levels in the technology independent phase; and, after technology mapping, the number of cells, the area, the worst negative slack (`WNS`) and the total negative slack (`TNS`). The slack is computed from the required time assigned to each netlist. The required time is shown in the first column of the table. The last row reports the normalized sum of the results without the netlist 11.

The improvement on the depth of the netlists is similar to the previous tables. However, there is a large improvement after technology mapping. The `WNS` is reduced by 40% and the area by 18%. The large difference between the results presented in this table and the previous ones is for several reasons. First, the netlist are even larger in area and similar in number of levels that the medium-size netlists of Table 2. BRCDEC is able to find good decompositions on large functions that contributes to improve the area and the delay. Moreover, the graph mapper takes profit of the sharing of common subexpressions reducing significantly the area of the netlist.

Analyzing the netlists individually, we can observe that BRCDEC obtains better results in all the circuits with the exception in netlists 5 and 6. The area is considerably reduced in all the netlist. Only netlists 9 and 11 have a worst result. The netlist 11 is a clear example of the selection of a wrong decision near the primary outputs on BiDec. The n-way decomposition is selected on the large functions that decreases significantly the size and the delay of the final circuit. However, the execution of BRCDEC takes around 10 hours to complete.

7. CONCLUSIONS

In this paper, a new application of the Boolean relations has been shown. The experimental results confirm that the n-way decomposition can obtain better solutions than bi-decomposition, mostly, on large functions. However, a high runtime is required and the execution is limited to medium-size netlists.

As a future work, we are planning to develop a window-based approach to run decomposition on larger netlists and perform the recursive decomposition on partial collapsed netlists. Moreover, we will explore the trade-off between the quality of the results and the required runtime to produce the solutions.

8. REFERENCES

[1] D. Bochmann, F. Dresig, and B. Steinbach, "A new decomposition method for multilevel circuit design," in *Proc. European Design Automation Conference (EURO-DAC)*, 1991, pp. 374–377.

[2] S. Yamashita, H. Sawada, and A. Nagoya, "New methods to find optimal non-disjoint bi-decompositions," in *Proc. ACM/IEEE Design Automation Conference*, 1998, pp. 59–68.

[3] C. Yang, M. Ciesielski, and V. Singhal, "BDS: A BDD-based logic optimization system," in *Proc. ACM/IEEE Design Automation Conference*, June 2000, pp. 92–97.

[4] A. Mishchenko, B. Steinbach, and M. Perkowski, "An algorithm for bi-decomposition of logic functions," in *Proc. ACM/IEEE Design Automation Conference*, June 2001, pp. 282–285.

[5] B. Steinback and A. Wereszczynski, "Synthesis of multi-level circuits using exor-gates," in *Proc. of Reed-Muller'95*, 1995, pp. 161–168.

[6] Y.-Y. Lian and Y.-L. Lin, "Layout-based logic decomposition for timing optimization," in *Proc. of Asia and South Pacific Design Automation Conference*, 1999, pp. 229–232.

[7] J. Cortadella, "Timing-driven logic bi-decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 675–685, June 2003.

[8] D. Kuck, *The Structure of Computers and Computation*. John Wiley & Sons, 1978.

[9] D. Baneres, J. Cortadella, and M. Kishinevsky, "A recursive paradigm to solve boolean relations," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 512–527, Apr. 2009.

[10] K. J. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1988, pp. 282–285.

[11] R. Rudell, "Logic synthesis for VLSI design," Ph.D. dissertation, UC Berkeley, Apr. 1989.

[12] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance-oriented technology mapping," in *Proc. 6th M.I.T. Conference on Advanced Research in VLSI*, Apr. 1990, pp. 79–97.

[13] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, pp. 519–526, 2005.