Graph-Based Transistor Network Generation Method for Supergate Design

Vinicius Neves Possani, *Student Member, IEEE*, Vinicius Callegaro, *Student Member, IEEE*, André I. Reis, *Senior Member, IEEE*, Renato P. Ribas, *Member, IEEE*, Felipe de Souza Marques, *Member, IEEE*, and Leomar Soares da Rosa, Jr., *Member, IEEE*

***Abstract*— Transistor network optimization represents an effective way of improving VLSI circuits. This paper proposes a novel method to automatically generate networks with minimal transistor count, starting from an irredundant sum-of-products expression as the input. The method is able to deliver both series–parallel (SP) and non-SP switch arrangements, improving speed, power dissipation, and area of CMOS gates. Experimental results demonstrate expected gains in comparison with related approaches.**

***Index Terms*— Automated synthesis, CMOS gates, digital circuit, switching theory, transistor network.**

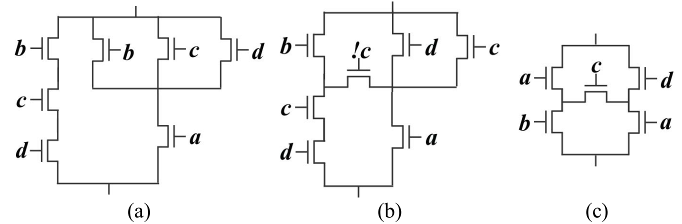


Fig. 1. Transistor networks corresponding to (1). (a) SP solution from factored form. (b) NSP from existing graph-based generation methods. (c) Optimum NSP solution.

I. Introduction

In VLSI digital design, the signal delay propagation, power dissipation, and area of circuits are strongly related to the number of transistors (switches) [1]–[3]. Hence, transistor arrangement optimization is of special interest when designing standard cell libraries and custom gates [4], [5]. Switch-based technologies, such as CMOS, FinFET [6], and carbon nanotubes [7], can take advantage of such an improvement. Therefore, efficient algorithms to automatically generate optimized transistor networks are quite useful for designing digital integrated circuits (ICs).

Several methods have been presented in the literature for generating and optimizing transistor networks. Most traditional solutions are based on factoring Boolean expressions, in which only series–parallel (SP) associations of transistors can be obtained from factored forms [8]–[11]. On the other hand, graph-based methods are able to find SP and also non-SP (NSP) arrangements with potential reduction in transistor count [12]–[15].

Despite the efforts of previous works, there is still a room for improving the generation of transistor networks. For instance, consider a given function represented by the following equation:

$$F = a \cdot b + a \cdot c + a \cdot d + b \cdot c \cdot d. \quad (1)$$

For this function, factorization methods are able to deliver the SP network shown in Fig. 1(a), comprising seven transistors. Existing graph-based methods, in turn, are able to provide the NSP solution shown in Fig. 1(b), also with seven transistors. However, the optimal arrangement composed of only five transistors, as shown in Fig. 1(c), is not found by any of these methods [8]–[15].

The proposed method starts from a sum-of-products (SOP) form F and produces a reduced transistor network. It comprises two main modules: 1) kernel identification and 2) network composition. The former aims to find efficient SP and NSP switch networks through graph structures called kernels. The latter receives the partial networks obtained from the first module and performs switch sharing, resulting in a single network representing F. Results have shown a significant reduction in transistor count when compared with other approaches [10]–[14]. Experiments have also demonstrated an improvement in performance, power dissipation, and area of CMOS gates as a consequence of such a device saving.

This paper is organized as follows. Section II reviews some fundamentals and definitions. Section III describes the novel method for transistor network generation. Section IV presents different execution modes of the proposed method considering the restriction of devices in series. Section V demonstrates the efficiency of the proposed approach by providing experimental results regarding transistor count, area estimation, gate performance, and power dissipation. Finally, the conclusion is drawn in Section VI.

Manuscript received March 19, 2014; revised December 18, 2014; accepted February 3, 2015. Date of publication March 20, 2015; date of current version January 19, 2016. This work was supported in part by the Brazilian funding agencies CAPES, CNPq and FAPERGS.

V. N. Possani, F. de Souza Marques, and L. S. da Rosa, Jr. are with the Development Technology Center, Federal University of Pelotas, Pelotas 960001-970, Brazil (e-mail: vini.possani@gmail.com; felipem@inf.ufpel.edu.br; leomarjr@inf.ufpel.edu.br).

V. Callegaro, A. I. Reis, and R. P. Ribas are with the Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre 90035-903, Brazil (e-mail: vcallegaro@inf.ufrgs.br; andreis@inf.ufrgs.br; rpribas@inf.ufrgs.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2015.2410764

1063-8210 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

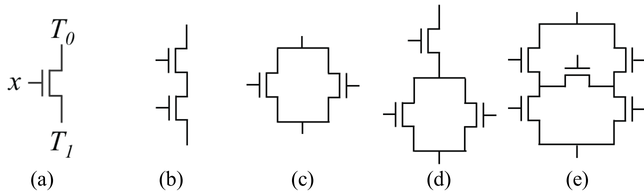


Fig. 2. Representation of (a) switch element (ideal MOS transistor), where x controls the connection between T and T_1 terminals, and associations. (b) Series. (c) Parallel. (d) SP. (e) NSP.

II. Definition and Preliminaries

A Boolean function $f(X)$ defined over the variable set $X = \{x_0, \ldots, x_{n-1}\}$ is a function defined as $f(X) : B^n \rightarrow B$, where $B = \{0, 1\}$ and $n = |X|$, i.e., the number of variables in X. The AND, OR, and NOT operations are denoted by $\cdot$, $+$, and !, respectively. A literal is a variable or its complement (e.g., x_i or $!x_i$), whereas a cube is a product of literals. An irredundant SOP (ISOP) is a SOP where neither a literal nor a cube can be removed without changing the represented function. Let f be a Boolean function given in ISOP form $F = c_1 + \cdots + c_m$, where m denotes the number of cubes in F. Considering that $i, j \leq m$, the union of the cubes c_i and c_j, denoted by $c_i \cup c_j$, returns the literals that belong to cube c_i or cube c_j. For instance, $a \cdot b \cup a \cdot c = \{a, b, c\}$. Notice that both positive and negative literals of the same variable can be returned, for example $a \cdot b \cup a \cdot c = \{a, !a, b, c\}$. An intersection of cubes c_i and c_j, denoted by $c_i \cap c_j$, returns literals that belong to both cubes, for example $a \cdot b \cap a \cdot c = a$. Notice that an empty cube can also be returned, e.g., $a \cdot b \cap d \cdot e = \emptyset$ [16].

A switch is a device composed by one control terminal and two contact terminals. The control terminal determines if there is a connection between the contact terminals, as shown in Fig. 2(a). In this sense, an ideal MOS transistor device acts as a switch. For this reason, the terms transistor and switch are used as synonymous in this paper. Moreover, series association of switches, as shown in Fig. 2(b), represents an AND operation, whereas parallel association, as seen in Fig. 2(c), corresponds to an OR operation. An SP switch network is obtained by iteratively connecting contact terminals in series and/or in parallel. An example of SP network is shown in Fig. 2(d). An NSP switch network is an arrangement that cannot be achieved by connecting terminal contacts in series and/or in parallel, as observed in Fig. 2(e). Notice that the function represented by a given switch network corresponds to the sum of all cubes associated to the paths between the contact terminals [17].

III. Switch Network Synthesis Method

The proposed method comprises two main modules: 1) the kernel identification and 2) the switch network composition. The former receives an ISOP F and identifies individual NSP and SP switch networks, representing subfunctions of f. The latter composes those networks into a single network by performing logic sharing. The provided output is an optimized switch network representing the target function f. The execution flow of the method is presented in Fig. 3.

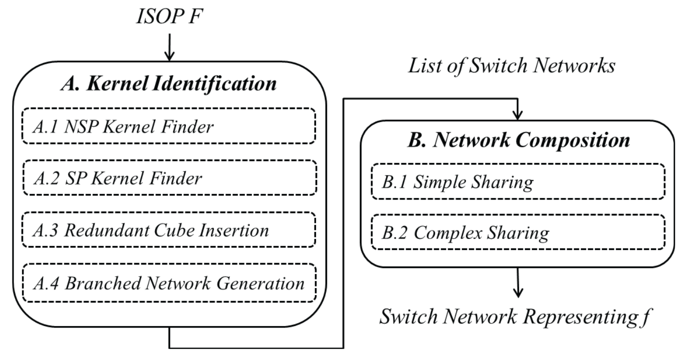


Fig. 3. Execution flow of the proposed method.

Algorithm 1 Pseudocode of the Kernel Identification Module

```
1:  kernelIdentification ( F )
2:    S ← Ø
3:    S ← S ∪ NSPKernelFinder( F )
4:    F1 ← removeImplementedCubes( F, S )
5:    S ← S ∪ SPKernelFinder( F1 )
6:    F2 ← removeImplementedCubes( F1, S )
7:    S ← S ∪ redundantCubeInsertion( F2 )
8:    F3 ← removeImplementedCubes( F2, S )
9:    S ← S ∪ branchedNetworkGeneration( F3 )
11:   return S
12: end
```

A. Kernel Identification

During the kernel identification module, an intermediate data structure called kernel is used to search for possible SP and NSP networks. A kernel of an ISOP F with m cubes is an undirected graph $G = (V, E)$, where vertices in $V = \{v_1, v_2, \ldots, v_m\}$ represent distinct cubes of F. An edge $e = (v_i, v_j) \in E$, $i \neq j$, exists if and only if $v_i \cap v_j \neq \emptyset$. Such edge e is labeled $v_i \cap v_j$. Using the kernel structure, it is possible to determine the relationship among cubes of F in order to perform logic sharing. This way, each step of the kernel identification module aims to extract kernels from F that leads to optimized switch count.

The kernel identification module is divided in four steps, as presented in Fig. 3 (left) and in Algorithm 1. Each step is responsible for finding switch networks representing subfunctions of the target function f. The NSP kernel finder step aims to obtain optimized NSP networks from an input ISOP F. When a switch network is found, the cubes used to achieve such network are removed from F. Such removal may lead to a simpler ISOP F_1.

The SP kernel finder step, in turn, searches for SP networks using as the input F_1. Similarly to the first step, the cubes of the found SP networks are removed from F_1, resulting F_2. Since the remaining cubes of F_2 were not useful to produce NSP or SP networks, redundant cubes are added into the kernels in order to find NSP arrangements with redundant paths. Therefore, the cubes leading to NSP networks with redundant paths are removed from F_2, resulting F_3. The last step produces branched switch networks, which comprises parallel paths corresponding to cubes from F_3 [18]. Finally, a list of switch networks is produced as output of the

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

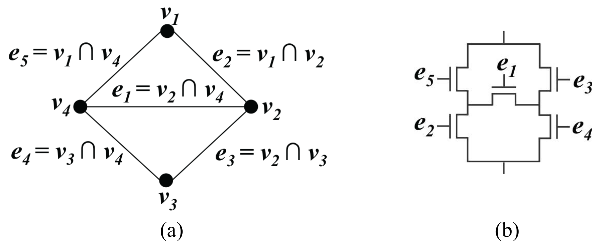


Fig. 4. (a) NSP kernel template. (b) Resulting switch network.

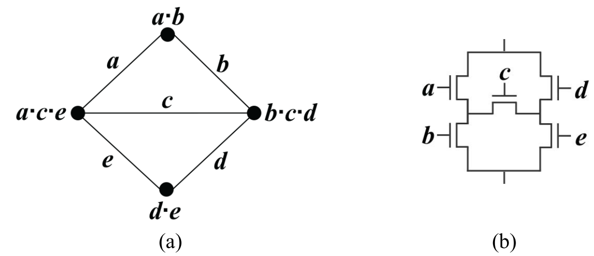


Fig. 5. (a) NSP kernel K_1, derived from (3). (b) Resulting switch network S_1.

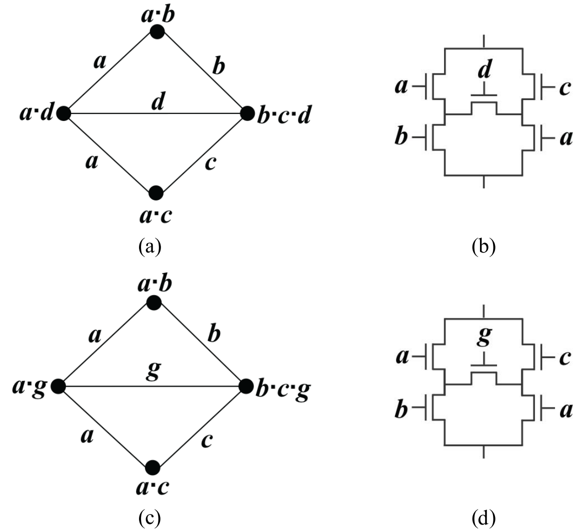


Fig. 6. NSP kernels (a) K_2 and (c) K_3, obtained from (4). Corresponding switch networks (b) S_2 and (d) S_3.

kernel identification module. Each step of this first module is detailed presented below.

1) Nonseries–Parallel Kernel Finder: Let f be a Boolean function given in ISOP form $F = c_1 + \cdots + c_m$, where m denotes the number of cubes in F. In order to identify NSP kernels, the combination of m cubes are taken four at a time, i.e., four-combination of cubes. The sum of such four cubes results in an ISOP H, which represents h that is a subfunction of f. A kernel with four vertices is obtained from H. To ensure that the generated kernel results in a NSP switch network, two rules must be checked.

Rule 1: Let E_υ be the set of edges connected to the vertex $\upsilon \in V$. For each cube (vertex) $\upsilon \in V$, all literals from υ must be shared through the edges $e \in E_\upsilon$. This rule is satisfied if and only if the following equation results the value 1:

$$\prod_{\upsilon \in V} \left(\left(\bigcup_{e \in E_\upsilon} e \right) = \upsilon \right). \tag{2}$$

Rule 2: The kernel obtained from H must be isomorphic to the graph shown in Fig. 4(a). Such a graph template is referred as NSP kernel.

An NSP kernel is mapped to a switch network by applying an edge swapping over three edges of the kernel. For instance, let us consider the generic NSP kernel shown in Fig. 4(a). To map this kernel to a network, the edge e_2 is moved to the place of e_4, e_4 is moved to the place of e_3, and e_3 is moved to the place of e_2. By applying such a reordering, it is possible to achieve the network shown in Fig. 4(b). The reordering procedure is necessary to ensure that each path of the switch network represents a cube from the subfunction h.

Example 1: Consider the following ISOP as the input to the NSP kernel finder step:

$$F = a \cdot b + a \cdot c \cdot e + d \cdot e + b \cdot c \cdot d. \tag{3}$$

The resulting kernel K_1 shown in Fig. 5(a), satisfies Rule 1 and Rule 2, and can be mapped by edge reordering to the switch network S_1, shown in Fig. 5(b). ■

Example 2: By combining cubes four at a time, the NSP kernel finder procedure can find more than one kernel per ISOP. For instance, consider the following equation:

$$F = a \cdot b + a \cdot c + c \cdot e + a \cdot d + b \cdot c \cdot d + a \cdot g + b \cdot c \cdot g. \tag{4}$$

For this ISOP, only two combinations of four cubes satisfy both Rule 1 and Rule 2, resulting in the NSP kernels K_2 and K_3 shown in Fig. 6(a) and (c), respectively. By applying the edge reordering procedure, these kernels are mapped to the switch networks S_2 and S_3 shown in Fig. 6(b) and (d), respectively. ■

Algorithm 2 Pseudocode of the NSP Kernel Finder Step

```
1:  NSPKernelFinder( F )
2:    C ← getCubeCombinations( F, 4 )
3:    S ← Ø
4:    for each combination c ∈ C do
5:      K ← obtainKernel( c )
6:      if ( K satisfies Rule 1 and Rule 2 ) then
7:        S ← S ∪ edgeReordering( K )
8:      end if
9:    end for
10:   return S
11: end
```

The pseudocode of the NSP kernel finder step is presented in Algorithm 2. Let C be the set of all possible four-combinations of cubes, generated by getCubeCombinations (line 2) procedure. Considering an ISOP F with m cubes, the getCubeCombinations has a time complexity of $O(m^4)$. Then, for each possible combination of four cubes, a kernel is obtained (line 5). The time complexity of the obtain kernel subroutine is $O(m^2 n)$, where n is the number of variables in F. Since only four cubes are considered at a time ($m = 4$), the time complexity can be simplified to $O(n)$.

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

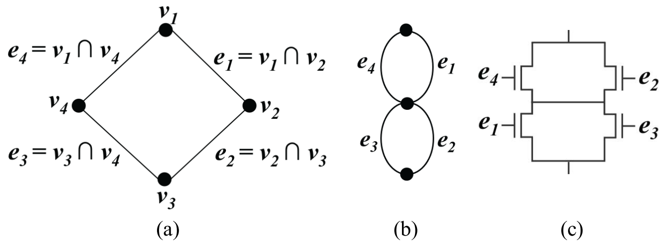


Fig. 7. (a) SP kernel template. (b) Auxiliary template graph. (c) Resulting switch network.

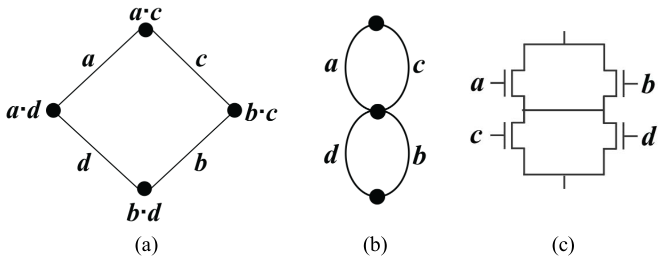


Fig. 8. (a) SP kernel K_4 derived from (5). (b) Auxiliary template graph A. (c) Switch network S_4 obtained after applying the edge reordering routine.

The time complexity to test Rule 1 is the same of the obtained kernel subroutine. A kernel satisfies Rule 2 if and only if the graph contains five edges. This test is done in constant time $O(1)$. In this sense, the time complexity to check both rules is bounded by $O(n)$. If the kernel satisfies both Rule 1 and Rule 2, the edge reordering subroutine is executed. Such a reordering is done in constant time. The resulting NSP switch network is then added to the found switch networks. This way, the time complexity of the NSP kernel finder procedure is bounded by $O(m^4)$.

2) Series–Parallel Kernel Finder: Let F_1 be an ISOP form that represents all the cubes of F that were not used to build switch networks in the NSP kernel finder step. To identify SP kernels, combination of m_1 cubes from F_1 are taken four at a time. A kernel with four vertices is then obtained. To ensure that the obtained kernel results in a valid SP network, Rule 1 and the following Rule 3 must be checked.

Rule 3: The obtained kernel must be isomorphic to the graph shown in Fig. 7(a). Such a graph template is referred as SP kernel.

Similarly to previous step, the SP kernel finder step must apply some transformations over the kernel in order to achieve a switch network. First, the kernel edges shown in Fig. 7(a) are mapped to an auxiliary template graph, as shown in Fig. 7(b). Afterward, a switch network is obtained by applying the edge reordering subroutine over the auxiliary template graph, as shown in Fig. 7(c).

Example 3: Consider the kernel K_4 shown in Fig. 8(a), obtained from the following equation:

$$F = a \cdot c + b \cdot c + b \cdot d + a \cdot d. \quad (5)$$

The edge labels are mapped from the kernel K_4 to the auxiliary graph A_4 shown in Fig. 8(b). Consequently, by applying the edge reordering over the graph A_4, the kernel K_4 is mapped to the switch network S_4 shown in Fig. 8(c).

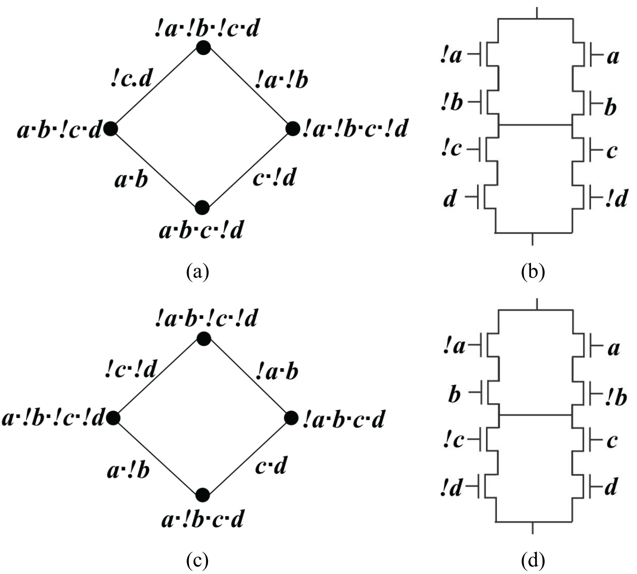


Fig. 9. SP kernels (a) K_5 and (c) K_6, obtained from (6). Corresponding switch networks (b) S_5 and (d) S_6.

Algorithm 3 Pseudocode of the SP Kernel Finder Step

```
1:  SPKernelFinder( F1 )
2:    C ← getCubeCombinations( F1, 4 )
3:    S ← Ø
4:    for each combination c ∈ C do
5:       K ← obtainKernel( c )
6:       if ( K satisfies Rule 1 and Rule 3 ) then
7:          S ← S ∪ edgeReordering( K )
8:       end if
9:    end for
10:   return S
11: end
```

Example 4: To demonstrate that multiple kernels can also be found during the SP kernel finder step, let us consider the following equation:

$$\begin{aligned} F = {} & !a \cdot !b \cdot !c \cdot d + !a \cdot !b \cdot c \cdot !d + !a \cdot b \cdot !c \cdot !d \\ & + !a \cdot b \cdot c \cdot d + a \cdot !b \cdot !c \cdot !d + a \cdot !b \cdot c \cdot d \\ & + a \cdot b \cdot !c \cdot d + a \cdot b \cdot c \cdot !d. \end{aligned} \quad (6)$$

For this function, the SP kernel finder procedure is able to find two SP kernels K_5 and K_6, shown in Fig. 9(a) and (c), respectively. Such kernels are remapped to the corresponding switch networks S_5 and S_6, as shown in Fig. 9(b) and (d), respectively.

The pseudocode of the SP kernel finder step is described in Algorithm 3. This pseudocode is quite similar to the NSP kernel finder one, presented in Algorithm 2. The main difference is that, in the line 6 of Algorithm 3, Rule 3 is checked instead of Rule 2. Basically, Rule 3 verifies if all vertices of the kernel have degree equals to two, as shown in Fig. 7(a). This test is done in constant time. This way, it is easy to see that the worst case time complexity of the SP kernel finder procedure is bounded by the cost of combining m_1 cubes from F_1, i.e., $O((m_1)^4)$.

3) Redundant Cube Insertion: In some cases, it is useful to build NSP arrangements with redundant cubes instead of using

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

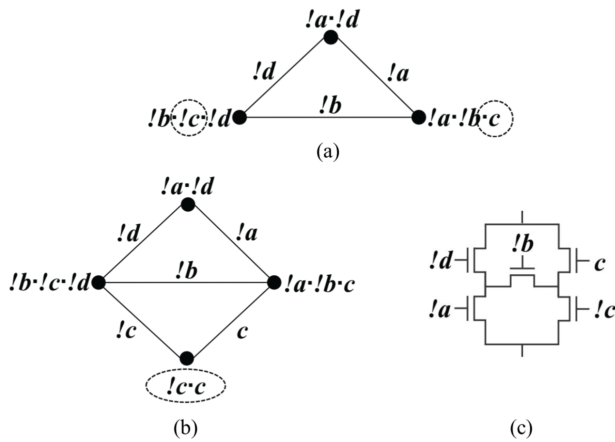


Fig. 10. (a) Graph G_1 obtained from (8). (b) Used to determine a NSP kernel K_7 with a redundant cube. (c) Resulting switch network S_7.

Algorithm 4 Pseudocode of the Redundant Cube Insertion Step

```
1:  redundantCubeInsertion( F2 )
2:    C ← getCubeCombinations( F2, 3 )
3:    S ← Ø
4:    for each combination c ∈ C do
5:      K ← obtainKernel( c )
6:      K ← insertRedundantCube( K )
7:      if ( K satisfies Rule 1 and Rule 2 ) then
8:        S ← S ∪ edgeReordering( K )
9:      end if
10:   end for
11:   return S
12: end
```

SP associations. Thus, when there still cubes not represented through NSP and SP networks, the redundant cube insertion step tries to build NSP kernels by combining remaining cubes with redundant cubes.

Let F be an ISOP representing the Boolean function f. A cube c is redundant if $F + c = f$. Consider a switch network representing an ISOP f. An implementation of a redundant cube c in such a network leads to a redundant logic path, i.e., the path does not contribute to the logic behavior of the network. Even though, redundant paths allow efficient logic sharing in NSP networks.

The redundant cube insertion step works over an ISOP F_2 representing the cubes that were not implemented by NSP and SP kernel finder steps. To obtain NSP kernels with redundant cubes, combinations of m_2 cubes are taken three at a time, where m_2 is the number of cubes in F_2. A kernel with three vertices is then obtained for each combination. Thus, a fourth cube (vertex) v_z is inserted into the kernel according to the following rule.

Rule 4: Let E_υ be the set of edges connected to the vertex $\upsilon \in V$. For each cube (vertex) $\upsilon \in V$, the literals from υ that were not shared through the edges $e \in E_\upsilon$ are inserted in v_z. Hence, the literals of the new vertex v_z are obtained by

$$v_z = \prod_{\upsilon \in V} \left(\upsilon - \bigcup_{e \in E_\upsilon} e \right) \tag{7}$$

where minus signal $(-)$ denotes relative complement. Therefore, after building the redundant cube v_z, Rule 1 and Rule 2 are applied over the resulting kernel in order to check if the cubes share all their literals through the edges.

Example 5: To demonstrate an NSP kernel with a redundant cube, let us consider the following equation:

$$F = !a \cdot !d + !a \cdot !b \cdot c + !b \cdot !c \cdot !d. \tag{8}$$

The graph G_1 obtained from (8) is shown in Fig. 10(a). This graph contains three edges, representing common literals between the cubes. Notice that the literals circled by dashed lines are not shared through any edge. In this sense, by applying Rule 4, these literals can be merged into a single vertex $v_z = \boldsymbol{!c \cdot c}$. By adding the vertex v_z into the graph, a valid NSP kernel K_7 is obtained, as presented in Fig. 10(b). As can be seen, v_z represents a redundant cube, i.e., $F + \boldsymbol{!c \cdot c} = f$. Hence, by applying the edge reordering routine over K_7, the method provides the optimized network S_7 comprising five switches, as shown in Fig. 10(c). Notice that the exact factoring for (8) comprises six literals, as shown in the following equation, resulting in a SP network with six switches:

$$F = (!d + c \cdot !b) \cdot (!a + !c \cdot !b). \tag{9}$$

Algorithm 4 presents the procedure to determine an NSP kernel with a redundant cube. There are two differences between this algorithm and the Algorithm 2 of the NSP kernel finder step. The first one is that, instead of generating four-combinations, Algorithm 4 generates three-combinations of cubes (line 2). The second difference is that, after obtaining the kernel from the selected cubes, the procedure inserts a redundant cube v_z into the kernel (line 6). Since only three cubes are considered at a time, the time complexity to build the redundant cube is $O(n_2)$, where n_2 is the number of variables in F_2. The time complexity of the redundant cube insertion step is bounded by the cost to perform cube combinations, i.e., $O((m_2)^3)$.

4) Branched Network Generation: Cubes from ISOP F are removed when a network implementation representing it is found. Even though previous steps are very efficient in finding logic sharing, there may still cubes not represented through any of the found networks. In this sense, the remaining cubes in F_3 are implemented as a single switch network. Therefore, the branched network generation step translates each remaining cube in F_3 to a branch of switches associate in series.

Example 6: To demonstrate how a branched network is obtained, consider the following equation:

$$F = a \cdot b \cdot c \cdot !d + !a \cdot b \cdot !c + a \cdot !b \cdot d. \tag{10}$$

In this case, there are three remaining cubes to be implemented. The obtained network is demonstrated in Fig. 11, where each literal in a cube was directly translated to a switch in the network.

Algorithm 5 presents the pseudocode of the branched network generation step. Basically, a switch network N starts empty and each cube is placed into the network. After placing

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

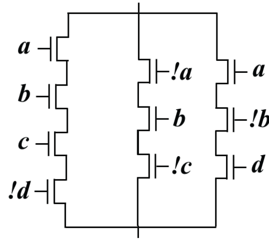


Fig. 11. Branched switch network obtained from (10).

Algorithm 5 Pseudocode of the Branched Network Generation Step

```
1: branchedNetworkGeneration( F3 )
2:   N ← Ø
3:   for each cube ∈ F3 do
4:     N ← placeCube( N, cube )
5:   end for
6:   return N
7: end
```

Algorithm 6 Pseudocode of the Network Composition Module

```
1: networkComposition( F, S )
2:   network ← Ø
3:   for each s ∈ S do
4:     network ← makeParallelAssociation( network, s )
5:     network ← simpleSwitchSharing( F, network )
6:     network ← complexSwitchSharing( F, network )
7:   end for
8:   return network
9: end
```

all cubes, the network is returned. The branched generation is a quite simple process with time complexity of $O(m_3 n_3)$, where m_3 is the number of the remaining cubes and n_3 is the number of variables in F_3.

B. Network Composition

The network composition module receives the function F and a list of partial switch networks S, generated during the kernel identification module. This module composes the networks from S in an iterative process by performing logic sharing among such networks. The target network starts empty and, for each network $s \in S$ a parallel association is performed together with simple and complex sharing strategies. The simple and the complex switch sharing are applied in order to remove redundant switches in the target network. The pseudocode of the network composition is presented in Algorithm 6. The makeParallelAssociation subroutine, in line 4, just places two networks in parallel. This way, this subroutine runs in constant time $O(1)$. The simple and the complex switch sharing steps are presented in the following sections *1) Simple Sharing* and *2) Complex Sharing* together with their respective time complexities.

1) Simple Sharing: The simple sharing step implements the edge sharing technique presented in [13]. Basically, the method traverses the switch network searching for equivalent switches, i.e., switches that are controlled by the same literal. The network is then restructured in such a way that one common node between equivalent switches is available. In some cases, the equivalent switches must be swapped in the networks in order to share a common node. When a common node between equivalent switches is available, only one switch is necessary, leading to a reduction in the number of switches.

Algorithm 7 Pseudocode of the Simple Sharing Step

```
1:  simpleSharing( F, network )
2:    do
3:      initialSwitchCount = getSwitchCount( network )
4:      E ← findEquivalentSwitches( network )
5:      if ( E ≠ ∅ ) then
6:        network ← switchSwapping( network, E )
7:        network ← switchSharing( network, E )
8:        network ← logicalEquivalenceChecking( F, network )
9:      end if
10:     finalSwitchCount = getSwitchCount( network )
11:   while ( initialSwitchCount > finalSwitchCount )
12:   return network
13: end
```

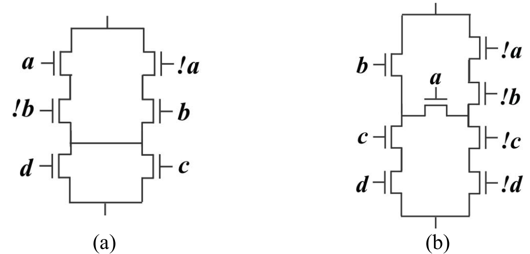


Fig. 12. Networks obtained from (11). (a) SP network S_9. (b) NSP network S_{10} with a redundant cube $!a \cdot !b \cdot a \cdot c \cdot d$.

After performing a switch sharing, the logic behavior of the network must be checked to ensure an accurate implementation of the target function. The switch sharing is accepted only if the logic behavior of the network is maintained. This optimization and validation process is applied iteratively over the network until there is no more feasible switch sharing to be applied.

A high level description of the simple sharing step is presented in Algorithm 7. Among all operations and subroutines needed to perform simple switch sharing, the highest time complexity is given by the logicalEquivalence Checking subroutine, in line 8. This procedure verify all logic paths of the network, requiring a time complexity of $O(2^{e/2})$, where e is the number of switches (edges) in the network. Thus, the simple sharing step is bounded by $O(2^{e/2})$.

Example 7: As an example of simple switch sharing, consider the following input ISOP:

$$F = !a \cdot !b \cdot !c \cdot !d + !a \cdot b \cdot d + !a \cdot b \cdot c + a \cdot !b \cdot d + a \cdot !b \cdot c + a \cdot b \cdot !c \cdot !d + b \cdot c \cdot d. \quad (11)$$

In this case, the SP kernel finder step was able to find the SP network S_9 shown in Fig. 12(a). Moreover, the redundant cube insertion step was able to find the NSP network S_{10} shown in Fig. 12(b). In order to compose a network corresponding to the given function described in (11), these partial switch networks S_9 and S_{10} are associated in parallel, as shown in Fig. 13(a).

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

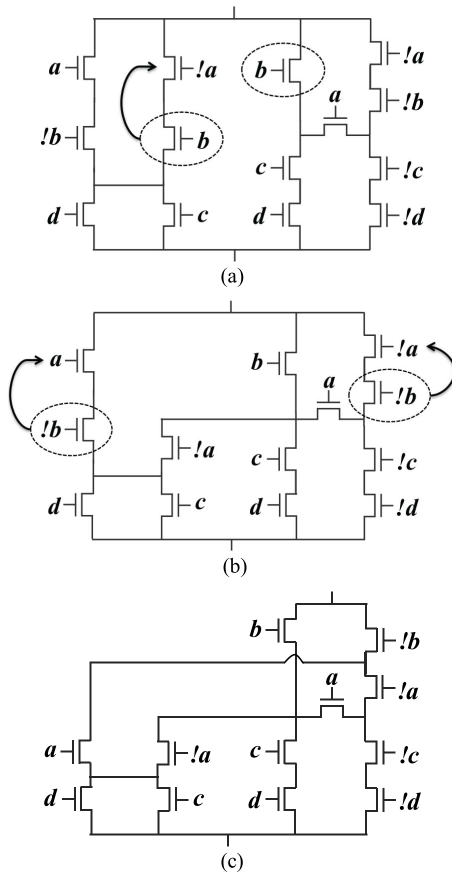


Fig. 13. (a) Parallel association of networks S_9 and S_{10}. (b) Intermediate network. (c) Final network to implement (11).

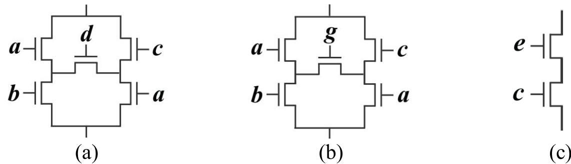


Fig. 14. Networks obtained from (4). (a) NSP network S_{11}. (b) NSP network S_{12}. (c) Branched network S_{13} implementing the remaining cube $\boldsymbol{e} \cdot \boldsymbol{c}$.

Notice that the network shown in Fig. 13(a) has some redundant switches, circled by dashed. Therefore, the simple sharing step is applied to remove such redundancies, resulting in the network shown in Fig. 13(b). The arrows in Fig. 13 indicate a swap between two switches. The optimization process is repeated until there are no more redundant switches in the network. Finally, the network shown in Fig. 13(c) represents the target function from (11). ■

Example 8: Let us reconsider Example 2, where the NSP kernel finder step was able to generate the networks S_{11} and S_{12} shown in Fig. 14(a) and (b), respectively. Observe that the cube $\boldsymbol{e} \cdot \boldsymbol{c}$ was not implemented in S_{11} and S_{12}. Hence, this cube was implemented as an independent branch S_{13} in the branched network generation step, as shown in Fig. 14(c). When the network composition starts, the first two networks, S_{11} and S_{12}, are associated in parallel as shown in Fig. 15(a). Then, by applying the simple switch sharing over S_{11} and S_{12},

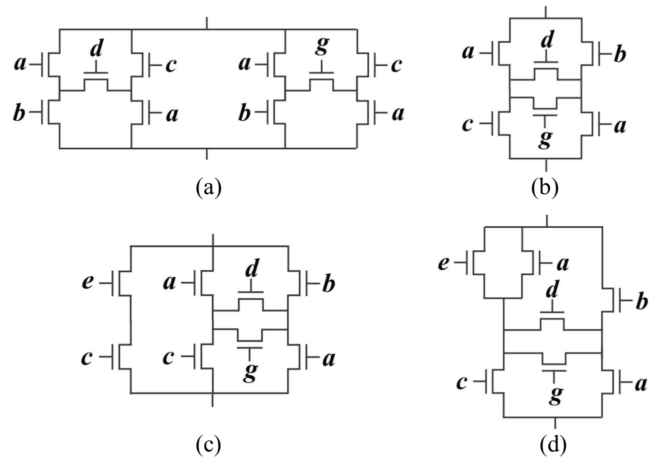


Fig. 15. (a) Parallel association of networks S_{11} and S_{12} obtained from Example 2. (b) Merged networks. (c) Parallel association of the branched network S_{13}. (d) Resulting switch network.

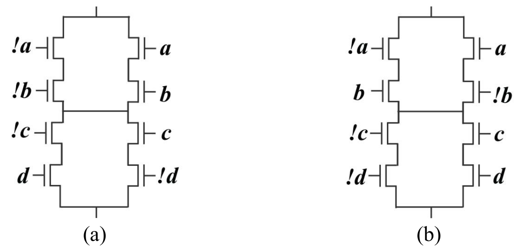


Fig. 16. Networks obtained from (6). SP networks (a) S_{14} and (b) S_{15}.

the equivalent switches are shared, as shown in Fig. 15(b). Thus, the branded network S_{13} of the cube $\boldsymbol{e} \cdot \boldsymbol{c}$ is inserted into the network, as shown in Fig. 15(c). Finally, by reapplying the switch sharing procedure, the final solution is found, as shown in Fig. 15(d). ■

Example 9: In order to demonstrate the potential of the simple sharing procedure even when applied over SP networks, let us revisit Example 4. In this case, the SP kernel finder step was able to generate two optimized switch networks S_{14} and S_{15} shown in Fig. 16(a) and (b), respectively. By arranging S_{14} and S_{15} in parallel, the switch network shown in Fig. 17(a) is obtained. Then, the simple sharing technique is applied to remove redundant switches from the network, resulting in the NSP solution shown in Fig. 17(b). Notice that the method is able to start from SP arrangements and achieve more optimized networks. ■

2) Complex Sharing: The complex sharing step receives a preprocessed network provided by the previous step and tries to perform additional optimizations. As mentioned in the simple sharing step, after finding equivalent switches, the procedure checks if the candidate switches have a common node that enables sharing. However, there are some cases where a common node is not directly found due to the position of the switches in the network. Hence, in order to improve the switch sharing, straightforward SP switch compressions are performed, as shown in Fig. 18(a) and (b), respectively. Then, simple switch sharing is applied over the compressed network.

Algorithm 8 presents the pseudocode of the complex sharing step. The main idea of the method is to execute the

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

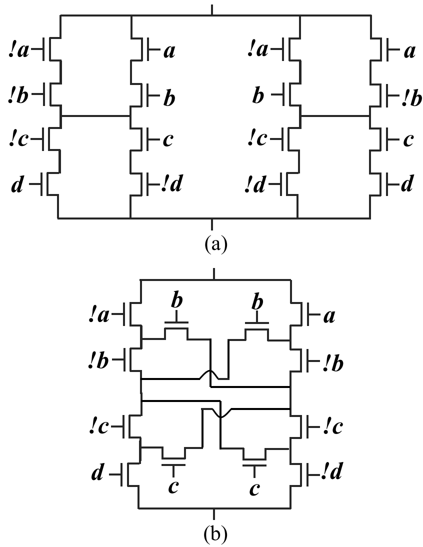


Fig. 17. (a) Parallel association of the switch networks S_{14} and S_{15}. (b) Optimized NSP switch network obtained by applying simple sharing.

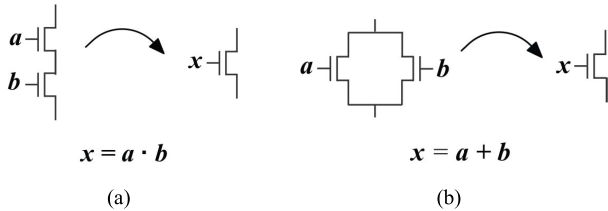


Fig. 18. (a) Series switch compression. (b) Parallel switch compression.

Algorithm 8 Pseudocode of the Complex Sharing Step

```
1: complexSharing( F, network )
2:   do
3:     SPCompression( network )
4:     initialSwitchCount = getSwitchCount( network )
5:     simpleSharing( F, network )
6:     finalSwitchCount = getSwitchCount( network )
7:   while ( initialSwitchCount > finalSwitchCount )
8:   SPExpansion( network )
9:   return network
10: end
```

SP compression and the simple sharing procedure in an iterative way, as presented in lines 3 and 5. One level of SP compression is applied in each step, allowing that multiple switches can be simultaneously swapped in the network. Consequently, it allows the sharing of multiple switches at the same time. At the end of this iterative process, the optimized network is expanded, as presented in line 8 of Algorithm 8. This is necessary to normalize the network, i.e., expand SP compressions in such a way that each switch is controlled by a single literal. Both the SP compression and SP expansion subroutines traverse the graph finding switches for compressing and expanding, respectively. The time complexity to traverse a switch network with c contact nodes and s switches is $O(c + s)$. In this sense, the time complexity of the complex sharing is bounded by the complexity of the simple sharing step, which is $O(2^{e/2})$.

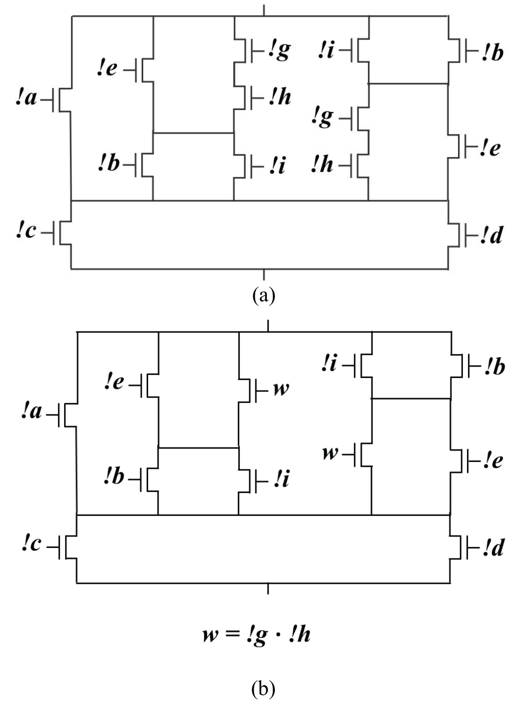


Fig. 19. (a) Switch network received as the input to the complex sharing step when processing the function described in (12). (b) Resulting network after performing series compressions.

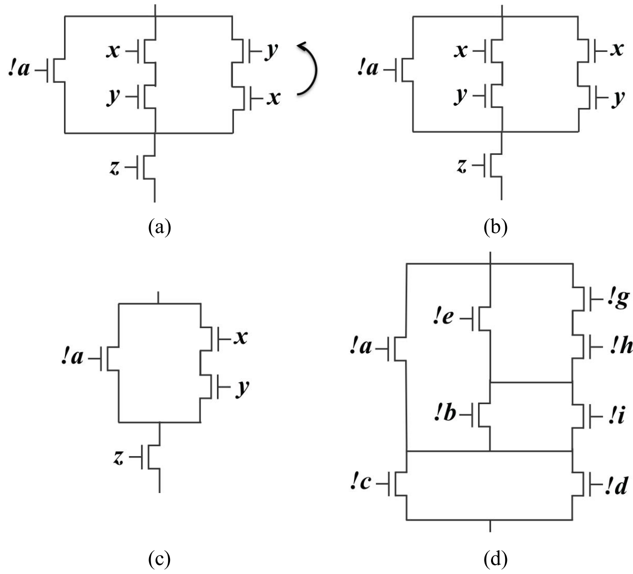
Example 11: Consider the following ISOP:

$$F = !a \cdot !c + !d \cdot !e \cdot !i + !a \cdot !d + !b \cdot !d \cdot !g \cdot !h + !b \cdot !c \cdot !g \cdot !h + !b \cdot !c \cdot !e + !d \cdot !i \cdot !g \cdot !h + !c \cdot !i \cdot !g \cdot !h + !c \cdot !e \cdot !i + !b \cdot !d \cdot !e. \quad (12)$$

After running the kernel identification module and performing the simple sharing step, the network shown in Fig. 19(a) is obtained. As can be seen, there are still some redundant switches in this network. The simple sharing step cannot remove such redundancies due to the position of switches. In this case, a common node between the redundant switches is not available. Moreover, none of these switches can be moved to other node without changing the logic behavior of the network. In this sense, a succession of SP compressions are performed. For instance, a series compression is applied over the switches $\boldsymbol{!g}$ and $\boldsymbol{!h}$, resulting in a single switch $\boldsymbol{w = !g \cdot !h}$, as shown in Fig. 19(b).

Afterward, a parallel switch compression is applied over the switches $\boldsymbol{!e}$ and $\boldsymbol{w}$ resulting in a single switch $\boldsymbol{x = !e + !g \cdot !h}$. The parallel compression is also applied over the switches $\boldsymbol{!b}$ and $\boldsymbol{!i}$, resulting in $\boldsymbol{y = !b + !i}$, and over switches $\boldsymbol{!c}$ and $\boldsymbol{!d}$, resulting in $\boldsymbol{z = !c + !d}$. The obtained network is shown in Fig. 20(a). A swap operation is applied over the compressed switches, resulting in the arrangement shown in Fig. 20(b). As can be seen, there is at least one common node between redundant switches. Hence, running the simple sharing procedure once more, it is possible to achieve the network shown in Fig. 20(c). Finally, an SP expansion is performed over the network, as shown in Fig. 20(d). ■

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.



$x = !e + !g \cdot !h$ $\quad y = !b + !i$ $\quad z = !c + !d$

Fig. 20. Obtained networks. (a) Performing parallel compressions. (b) Swapping x and y. (c) After sharing the equivalent switches x and y. (d) Final network after applying the SP expansion.

IV. Transistor Stack Bounding

Switch networks can be exploited by switch-based technologies, which present some restrictions or guidelines to be followed by designers. For example, in the conventional CMOS design technology, the maximum number of stacked transistors is usually limited to four. Such restriction is done in order to avoid performance degradation. Notice that there is a lower bound on the stacked transistors in switch networks. This lower bound corresponds to the minimum decision chain (MDC) property of the represented Boolean functions [19]. In this sense, an interesting feature to control (or to limit) the number of stacked transistors was included in our method. The method can operate in two execution modes, bounded and unbounded, as described below.

A. Bounded Mode

In this execution mode, a bound variable is used as reference to control the maximum number of transistors in series. The bound value must be equal or greater than the number of literals of the smallest cube from F, i.e., the maximum number of literals in a single cube. When the method is running in the bounded mode, the kernel identification module accepts only switch networks in which maximum stacked transistors do not exceed the bound value. Hence, the networks satisfying such a bound are added to the list S of found networks. This control is also performed during the network composition module when applying switch sharing, since it can increase the transistor stack.

B. Unbounded Mode

When running in the unbounded mode, there is no restriction of transistor stacking, i.e., the bound variable is not considered. Basically, just the total transistor count of the network is taken as metric cost. Hence, there are cases that the networks generated through the unbounded mode result fewer transistors when compared with bounded solutions. Moreover, these different modes are quite useful to explore the tradeoff between circuit area and performance.

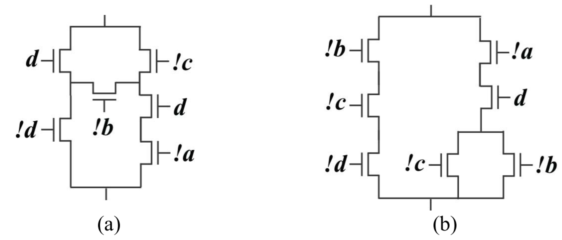


Fig. 21. Two possible transistor networks generated by the proposed method to implement the function described in (13), considering both execution modes. (a) Unbounded. (b) Bounded.

Example 12: In order to demonstrate these two execution modes, consider the following equation:

$$F = \ !a \cdot !b \cdot d + !a \cdot !c \cdot d + !b \cdot !c \cdot !d. \qquad (13)$$

The network shown in Fig. 21(a) is obtained when running the method in the unbounded mode. Notice that, in this network, the size of the transistor stack $!a \cdot d \cdot !b \cdot d$ is >3, which is the number of literals of the smallest cube from (13). When running the method in the bounded mode, it is possible to ensure transistor stacks with at most three devices, as shown in Fig. 21(b). On one hand, the bounded network presents an overhead of one transistor in comparison with the unbounded solution. On the other hand, the bounded solution has smallest transistor stacks. In this sense, one can consider the bounded solution when targeting performance or the unbounded solution for smaller area. ■

V. Experimental Results

For evaluation and validation, the proposed method was applied over different sets of representative functions in order to provide a fair comparison with other available solutions [10]–[14]. Four different set of functions were considered: 1) the set of 4-input P-class of functions; 2) a set of handcrafted networks that do not present transistors in SP associations [20]; 3) a given function with 11 variables as a more complex case study; and 4) the functions and transistor networks described in the Ninomiya's catalog [21].

A. Transistor Count Evaluation

The first experiment was carried out over Boolean functions up to four variables, representing 65 536 functions. These functions were grouped into a set of equivalent classes by considering input permutation called, herein, 4-input P-class. This set comprises 3982 representative functions. The corresponding CMOS gate for each function of this set was built, by generating both the pull-up and pull-down networks through the methods in evaluation. Table I shows the results obtained considering the total transistor count used to build all gates, including required input inverters. Notice that, only input inverters are allowed.

The second experiment was carried out over a set of 53 functions obtained from handmade networks where there

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

TABLE I
TRANSISTOR COUNT FOR THE 4-INPUT P-CLASS FUNCTIONS

	[11]	[12]	[13]	[14]	**Bounded**	**Unbounded**
Transistor count	102,668	103,049	96,804	97,174	**96,824**	**95,595**

TABLE II
TRANSISTOR COUNT FOR THE 53 HANDMADE NETWORKS [20]

	[20] (optimum)	[11]	[12]	[13]	[14]	**Bounded**	**Unbounded**
Transistor count	356	487	503	516	543	**383**	**359**

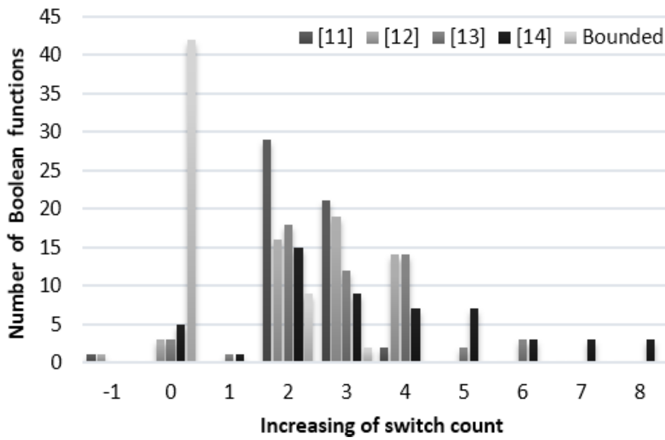


Fig. 22. Histogram comparing the proposed method (unbounded) with other approaches.

are neither series nor parallel associations [20]. These networks have at most four transistors in series, being suitable for CMOS gate implementation. It represents a worthy benchmark set taken as reference to evaluate the proposed approach and other related methods [11]–[14]. The results are summarized in Table II.

Fig. 22 shows the distribution of gains and losses when synthesizing transistor networks for the set of 53 handmade functions. In this histogram, horizontal axis corresponds to the transistor count overhead in respect to the proposed method (unbounded), used herein as reference. The vertical axis relates the number of functions for each increase on transistor count. As can be seen, our unbounded method was able to reduce up to eight transistors in some cases. In general, the gains are around two to three transistors per network. Although there is an increasing in transistor count, the proposed method (bounded) is near to the optimum transistor count [20].

A more complex case study is presented using an 11-input function, comprising 99 literals as follows:

$$\begin{aligned} F = {} & a\cdot i + c\cdot k + b\cdot d\cdot i + b\cdot m\cdot k + a\cdot g\cdot j + b\cdot e\cdot j \\ & + c\cdot h\cdot j + c\cdot m\cdot d\cdot i + b\cdot e\cdot g\cdot i + c\cdot h\cdot g\cdot i \\ & + a\cdot d\cdot m\cdot k + a\cdot g\cdot h\cdot k + b\cdot e\cdot h\cdot k + b\cdot d\cdot g\cdot j \\ & + a\cdot d\cdot e\cdot j + c\cdot m\cdot e\cdot j + b\cdot m\cdot h\cdot j \\ & + c\cdot h\cdot e\cdot d\cdot i + c\cdot m\cdot e\cdot g\cdot i + b\cdot m\cdot h\cdot g\cdot i \\ & + a\cdot g\cdot e\cdot m\cdot k + b\cdot d\cdot g\cdot h\cdot k + a\cdot d\cdot e\cdot h\cdot k \\ & + c\cdot m\cdot d\cdot g\cdot j + a\cdot d\cdot m\cdot h\cdot j. \end{aligned} \quad (14)$$

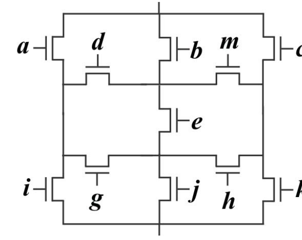


Fig. 23. Transistor network related to (14), provided by our method.

TABLE III
TRANSISTOR COUNT USED TO IMPLEMENT (14)

	[11]	[12]	[13]	[14]	**Bounded**	**Unbounded**
Transistor count	31	31	25	22	**11**	**11**

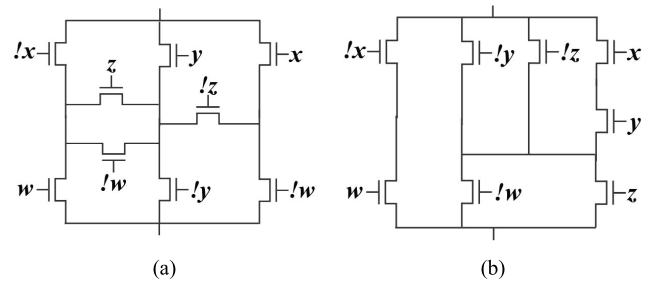


Fig. 24. Network related to (15). (a) Solution provided in the Ninomiya's catalog–function *N12′* [21]. (b) Arrangement delivered by our method.

In this case, the proposed method provides the transistor network presented in Fig. 23. This is the optimum solution with only 11 transistors. Table III presents the transistor count obtained from other methods, showing that our approach is able to deliver a network with at least 50% of reduction.

Finally, we have evaluated the proposed method using the set of 402 functions/networks presented in the Ninomiya's catalog [21], also adopted as benchmarking in other works. Kagaris and Haniotakis [14] and Tanaka and Kambayashi [22] claim that their methods have obtained networks with the same number of transistors as presented in this catalog, excepting for function *N58′* where they achieved a reduction of one transistor. Our method is not capable to achieve the best solution for all functions provided in the catalog. However, in this experiment, our contribution was to achieve a reduction of one transistor for the function *N12′*, represented in the catalog by the following equation:

$$F = w\cdot !x + !w\cdot !y + !w\cdot !z + !y\cdot z + x\cdot y\cdot z. \quad (15)$$

The Ninomiya's network for this function is shown in Fig. 24(a), whereas our solution with one transistor saving is shown in Fig. 24(b).

Moreover, we observed that the function *N54* from this catalog presents an inconsistency between the given Boolean expression

$$\begin{aligned} F = {} & w\cdot x\cdot !y + w\cdot x\cdot z + w\cdot !x\cdot y\cdot z \\ & + !w\cdot !x\cdot y\cdot z \end{aligned} \quad (16)$$

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

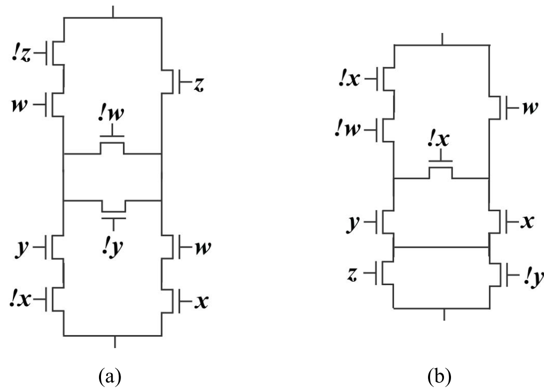


Fig. 25. (a) Network provided in the Ninomiya's catalog [21] for function *N54*, which is not logically equivalent to the catalog expression represented in (16). (b) Arrangement delivered by our method for this expression.

and the transistor network presented, shown in Fig. 25(a), said to be logically equivalent. The right solution, delivered by our method, is presented in Fig. 25(b).

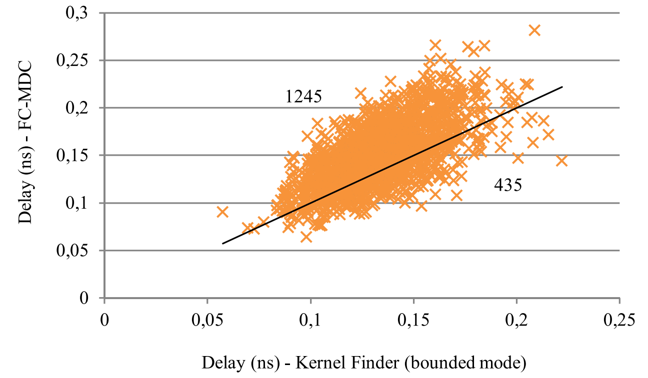
B. Performance and Area Evaluation

In terms of electrical and physical characteristics of CMOS gates, the reduction in the number of transistors may lead to improvements on speed, power dissipation, and area. However, there are other parameters that impact circuit quality such as transistor sizing, layout compaction, and waveform of input stimuli.

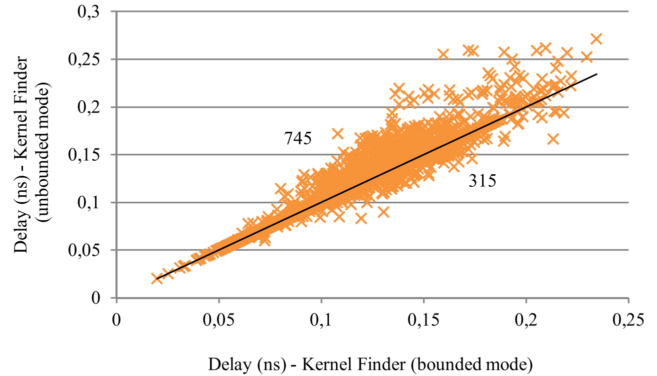
The experiments were carried out over the 4-inputs P-class set of functions. Then, three libraries were built, being the gates generated by applying the exact factorization method Functional Composition (FC)-MDC [10], as well as the kernel finder, bounded, and unbounded modes, proposed in this paper. Each library comprises 3982 gates. Notice that the network can be built considering the lower bound of transistors in stack for a given Boolean function [19]. In this sense, the FC-MDC and the bounded cell libraries were generated considering such lower bound. It is known that the longest transistor path is the main responsible for the worst case of delay propagation through the gate.

Electrical characterization of the libraries were carried out using the Cadence Encounter Library Characterizer tool, considering the 32-nm CMOS predictive technology model typical parameters [23] and the nominal power supply voltage of 1 V. First, transistors were sized according to the logical effort method [24] that considers the transistor stacking in the network. The channel length of transistors is 32 nm. The nMOS transistor width is 64 nm while the pMOS transistor width is defined using the PN ratio equals to two, in the inverter gate used as reference for the logical effort method. The input slopes and output load applied in the characterization process were defined considering the usual fan-out four.

Gate delay is mainly affected by the maximum stacked transistors in a network. Considering two networks with the same transistor stack, the network comprising fewer transistors usually has a better performance. In this sense, performance improvements can be observed in the comparative analysis



(a)



(b)

Fig. 26. Delay evaluation of networks, comparing the bounded execution mode to (a) exact factorization generation (FC-MDC) [10] and (b) unbounded mode generation.

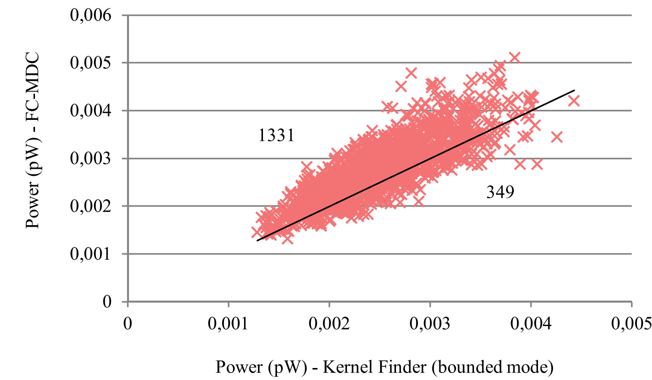
presented in Fig. 26. When comparing the bounded and the FC-MDC methods, only the cases where the proposed method reduced transistor count were considered. This analysis corresponds to 1680 gates, as shown in Fig. 26(a). The comparison between bounded and unbounded modes is shown in Fig. 26(b), where 2922 gates present the same topology and, consequently, the same behavior.

A decrease in device count tends to present a significant reduction in gate power dissipation. Fewer transistors in the gate represent less capacitance to be charged during the signals switching. The power reduction analysis comparing the bounded and the FC-MDC methods is shown in Fig. 27(a). Only the cases where the proposed method reduced the number of transistors is considered. The results confirmed that reducing transistor count leads to a power dissipation reduction.

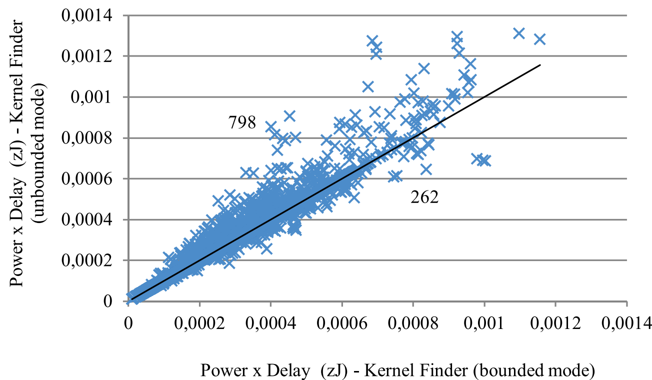
A power-delay analysis was performed comparing bounded and unbounded execution modes. The power-delay product evaluation is shown in Fig. 27(b). This analysis corresponds to the dynamic power component. The short-circuit component was not considered, since it usually represents $<10\%$ of the total power dissipation in well-designed circuits.

In terms of physical area evaluation, it is intuitive to expect area saving when the number of devices is reduced. One could think in building carefully the layout of a set of networks. However, such a task is impractical and may result even in an inconclusive analysis due to many other factors involved.

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.



(a)



(b)

Fig. 27. (a) Power dissipation analysis of the networks provided by bounded mode and exact factorization methods (FC-MDC) [10]. (b) Power-delay product of networks generated by bounded and unbounded modes.

The network layout dimensions can be estimated considering the usual standard cell template, where pMOS transistors of the pull-up network are placed over nMOS transistors of the pull-down plane. Such a transistor placement is made side-by-side, in line, exploring layout techniques like Euler paths [25]. The Euler path analysis gives a good idea of layout length, whereas the layout height can be estimated considering the power lines, the P and N active areas, and the signals wire congesting. Notice that the layout compaction is a very hard and handmade time consuming task not explored in this paper. The Euler path (and eventual active area breaks) and the signal routing were extracted for each generated network. An area estimative evaluation, considering only cases that the proposed method reduced transistor count, is shown in Fig. 28. Such cases comprise 1680 gates. For 254 gates, an area increase was observed due to breaks insertion in order to match Euler paths. For 1186 gates, an area reduction was observed, demonstrating the tendency of the circuit area saving as a result of transistor count reduction.

C. Execution Time

The total execution time of the proposed method to generate the networks described on all the experiments presented in this paper was 1.2 s. The platform was an Intel Core i5 processor at 2.8 GHz with 4 GB of RAM. It demonstrates the feasibility of

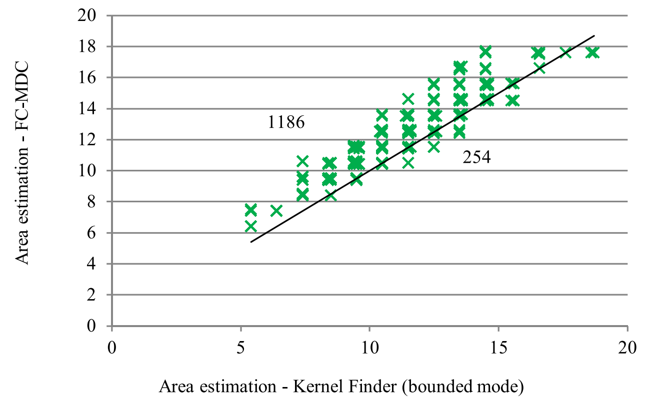


Fig. 28. Comparison between the networks provided by the bounded mode and the exact factorization generation (FC-MDC) [10] methods in terms of area estimation.

the proposed method to generate optimized transistor network, increasing design quality.

VI. CONCLUSION

This paper described an efficient graph-based method to generate optimized transistor (switch) networks. Our approach generates more general arrangements than the usual SP associations. Experimental results demonstrated a significant reduction in the number of transistor needed to implement logic networks, when compared with the ones generated by existing related approaches. It is known that the transistor count minimization in CMOS gates may improve the performance, power dissipation, and area of digital ICs. In a general point-of-view, the proposed method produces efficient switch arrangements quite useful to be explored by different IC technologies based on switch theory.

ACKNOWLEDGMENT

The authors would like to thank L. S. Puricelli, C. S. Nunes, and F. S. Marranghello for their electrical characterization support and valuable discussions.

REFERENCES

[1] Y.-T. Lai, Y.-C. Jiang, and H.-M. Chu, "BDD decomposition for mixed CMOS/PTL logic circuit synthesis," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 6. May 2005, pp. 5649–5652.
[2] H. Al-Hertani, D. Al-Khalili, and C. Rozon, "Accurate total static leakage current estimation in transistor stacks," in *Proc. IEEE Int. Conf. Comput. Syst. Appl.*, Mar. 2006, pp. 262–265.
[3] T. J. Thorp, G. S. Yee, and C. M. Sechen, "Design and synthesis of dynamic circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 1, pp. 141–149, Feb. 2003.
[4] A. I. Reis and O. C. Andersen, "Library sizing," U.S. Patent 8 015 517, Jun. 5, 2009.
[5] R. Roy, D. Bhattacharya, and V. Boppana, "Transistor-level optimization of digital designs with flex cells," *Computer*, vol. 38, no. 2, pp. 53–61, Feb. 2005.
[6] M. Rostami and K. Mohanram, "Dual-v_{th} independent-gate FinFETs for low power logic circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 3, pp. 337–349, Mar. 2011.
[7] M. H. Ben-Jamaa, K. Mohanram, and G. De Micheli, "An efficient gate library for ambipolar CNTFET logic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 2, pp. 242–255, Feb. 2011.
[8] M. C. Golumbic, A. Mintz, and U. Rotics, "An improvement on the complexity of factoring read-once Boolean functions," *Discrete Appl. Math.*, vol. 156, no. 10, pp. 1633–1636, May 2008.

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

[9] E. M. Sentovich *et al.*, "SIS: A system for sequential circuit synthesis," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/ERL M92/41, May 1992.
[10] M. G. A. Martins, V. Callegaro, L. Machado, R. P. Ribas, and A. I. Reis, "Functional composition and applications," in *Int. Workshop Logic Synthesis Tech. Dig. (IWLS)*, Jun. 2012, pp. 1–8. [Online]. Available: http://www.inf.ufrgs.br/logics/
[11] M. G. A. Martins, L. S. da Rosa, Jr., A. B. Rasmussen, R. P. Ribas, and A. I. Reis, "Boolean factoring with multi-objective goals," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2010, pp. 229–234.
[12] L. S. da Rosa, Jr., F. S. Marques, F. R. Schneider, R. P. Ribas, and A. I. Reis, "A comparative study of CMOS gates with minimum transistor stacks," in *Proc. 20th Annu. Conf. Integr. Circuits Syst. Design (SBCCI)*, Sep. 2007, pp. 93–98.
[13] V. N. Possani, R. S. de Souza, J. S. Domingues, Jr., L. V. Agostini, F. S. Marques, and L. S. da Rosa, Jr., "Optimizing transistor networks using a graph-based technique," *J. Analog Integr. Circuits Signal Process.*, vol. 73, no. 3, pp. 841–850, Dec. 2012.
[14] D. Kagaris and T. Haniotakis, "A methodology for transistor-efficient supergate design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 4, pp. 488–492, Apr. 2007.
[15] J. Zhu and M. Abd-El-Barr, "On the optimization of MOS circuits," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 40, no. 6, pp. 412–422, Jun. 1993.
[16] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer, 1984.
[17] T. Sasao, *Switching Theory for Logic Synthesis*. New York, NY, USA: Springer-Verlag, 1999.
[18] C. Piguet, J. Zahnd, A. Stauffer, and M. Bertarionne, "A metal-oriented layout structure for CMOS logic," *IEEE J. Solid-State Circuits*, vol. 19, no. 3, pp. 425–436, Jun. 1984.
[19] M. G. A. Martins, V. Callegaro, R. P. Ribas, and A. I. Reis, "Efficient method to compute minimum decision chains of Boolean functions," in *Proc. 21st Ed. Great Lakes Symp. VLSI (GLSVLSI)*, May 2011, pp. 419–422.
[20] Federal Univ. Rio Grande do Sul, Logics Lab. (Oct. 2012). *Catalog of 53 Handmade Optimum Switch Networks*. [Online]. Available: http://www.inf.ufrgs.br/logics/docman/53_NSP_Catalog.pdf
[21] M. A. Harrison, *Introduction to Switching and Automata Theory*. New York, NY, USA: McGraw-Hill, 1965, pp. 408–472.
[22] K. Tanaka and Y. Kambayashi, "Transduction method for design of logic cell structure," in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2004, pp. 600–603.
[23] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Trans. Electron Devices*, vol. 53, no. 11, pp. 2816–2823, Nov. 2006. [Online]. Available: http://ptm.asu.edu/
[24] I. E. Sutherland, R. F. Sproull, and D. F. Harris, *Logical Effort: Designing Fast CMOS Circuits*. San Mateo, CA, USA: Morgan Kaufmann, 1999.
[25] T. Uehara and W. M. Vancleemput, "Optimal layout of CMOS functional arrays," *IEEE Trans. Comput.*, vol. C-30, no. 5, pp. 305–312, May 1981.

Vinicius Neves Possani (S'13) received the B.S. and M.S. degrees in computer science from the Federal University of Pelotas (UFPEL), Pelotas, Brazil, in 2013 and 2015, respectively. He is currently pursuing the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil.

He was a Substitute Professor at UFPEL in 2014. His current research interests include logic synthesis, automatic generation, optimization of transistor (switch) networks, including new devices as double-gate FinFET transistor and emerging technologies, algorithms for electronic design automation, and technology mapping.

Vinicius Callegaro (S'14) received the B.S. and M.S. degrees in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2010 and 2012, respectively, where he is currently pursuing the Ph.D. degree in computer science.

His current research interests include logic synthesis methods for CMOS and emerging technologies, and automatic generation of standard cell libraries.

Mr. Callegaro received a best paper award at the Symposium on Integrated Circuit and Systems Design Conference in 2013.

André I. Reis (M'99–SM'05) received the B.S. degree in electrical engineering and the M.S. degree in computer science from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1991 and 1993, respectively, and the Ph.D. degree in automatic and microelectronics systems from the University of Montpellier II, Montpellier, France, in 1998.

He has been a Professor with the Department of Computer Science, UFRGS, since 2000. He was a Visiting Researcher with the University of Minnesota, Minneapolis, MN, USA, from 2004 to 2005. He was the Chief Scientist with Nangate A/S, Herlev, Denmark, from 2006 to 2009, where he has been a member of the Technical Advisory Board since 2005. He has successfully coordinated UFRGS participation in the Synaptic FP7 European Project Consortium. Furthermore, he is a Thesis Advisor for the M.S. and Ph.D. programs in computer science and a Co-Founder and Thesis Advisor for the master's and Ph.D. program in microelectronics at UFRGS. He holds 10 granted U.S. patents, and over 150 academic publications. He has co-authored books on digital circuit design.

Prof. Reis is a member of the Brazilian Microelectronics Society, the Brazilian Computer Society, and the Association for Computing Machinery. He received the best paper award at the International Federation for Information Processing International Conference on Very Large Scale Integration in 1997 and the Symposium on Integrated Circuit and Systems Design Conference in 2013. He is serving as the General Chair for the 24th International Workshop on Logic and Synthesis.

Renato P. Ribas (M'12) received the B.S. degree in electrical engineering from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1991, the M.S. degree in electrical engineering from the University of Campinas, Campinas, Brazil, in 1994, and the Ph.D. degree in microelectronics from the Institut National Polytechnique de Grenoble, Grenoble, France, in 1998.

He has been a Professor with the Department of Applied Informatics, Informatics Institute, UFRGS, since 2000. He was the Chief Scientist with Nangate A/S, Herlev, Denmark, in 2005, and a Post-Doctoral Researcher with the University of British Columbia, Vancouver, BC, Canada, in 2010. He has authored or co-authored over 140 technical papers. He holds a U.S. patent. His current research interests include VLSI system and circuit design, computer-aided design development, and new technologies for digital IC design.

Mr. Ribas was the President of the Brazilian Microelectronics Society and a Co-ordinator of the Computer Engineering course at UFRGS.

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.

Felipe de Souza Marques (M'10) received the B.S. degree in computer science from the Federal University of Pelotas, Pelotas, Brazil, in 2000, and the M.S. and Ph.D. degrees in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2003 and 2008, respectively.

He is currently a Professor with the Technology Development Center, Federal University of Pelotas. His current research interests include electronic design automation, logic and physical synthesis, focusing on technology mapping, automatic cell generation and optimization of circuits and switches networks.

Prof. Marques is a member of the IEEE Computer Society, the Association for Computing Machinery, the Brazilian Computer Society, and the Brazilian Microelectronics Society.

Leomar Soares da Rosa, Jr. (M'11) received the B.S. degree in computer science from the Federal University of Pelotas, Pelotas, Brazil, in 2001, and the M.S. degree in computer science and the Ph.D. degree in microelectronics from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2003 and 2008, respectively.

He is currently a Professor with the Technology Development Center, Federal University of Pelotas. His current research interests include electronic design automation, logic synthesis, technology mapping, automatic generation, and optimization of circuits and switches networks.

He is a member of the IEEE Computer Society, the Association for Computing Machinery, the Brazilian Computer Society, and the Brazilian Microelectronics Society.

Authorized licensed use limited to: Univ of Calif San Diego. Downloaded on November 27,2023 at 23:59:09 UTC from IEEE Xplore. Restrictions apply.