Realization of Logic Functions Using Switching Lattices Under a Delay Constraint

Levent Aksoy, Nihat Akkan, Herman Sedef, and Mustafa Altun

***Abstract*—Switching lattices, consisting of four-terminal switches, present an alternative structure for the realization of Boolean logic functions. Although promising algorithms have been introduced to find a realization of a logic function using a switching lattice with the fewest number of four-terminal switches, the delay of a switching lattice has not been examined yet. In this article, we generate a switching lattice using a recently proposed CMOS-compatible four-terminal device model and formulate the delay of a path in a switching lattice. It is observed that the delay of a design realizing a logic function on a switching lattice heavily depends on the number of four-terminal switches in the critical path. With this motivation, we introduce optimization algorithms, called PHAEDRA and TROADES, that can find the realization of a logic function on a switching lattice with the fewest number of switches under a delay constraint given in terms of the number of switches in the critical path. While PHAEDRA is a dichotomic search algorithm that can obtain solutions with a small number of switches on small size logic functions, TROADES is a divide-and-conquer method that can find a solution using less computational effort and can easily handle larger size logic functions with respect to PHAEDRA. Experimental results show that the proposed algorithms can reduce the delay of a lattice realization of a logic function significantly at a cost of an increase in the number of switches. They can explore alternative lattice realizations of a logic function by changing the delay constraint, enabling a designer to choose the one that fits best in an application.**

***Index Terms*—emerging technologies, four-terminal switch, switching lattice, logic synthesis, satisfiability, Elmore delay, binary search, divide and conquer.**

I. Introduction

As the Moore's law [1] is reaching its limits, new structures and architectures for the nano-electronic computation have been introduced [2]–[5]. As shown in Fig. 1(a), a four-terminal switch, developed for the cross-points of nanoarrays, connects all its terminals if its control input x has the logic value 1, otherwise its terminals are disconnected. A switching lattice is formed as a network of four-terminal switches where each switch is connected to its horizontal and vertical neighbors. Fig. 1(b) presents the 4×2 lattice where $x_1 \dots x_8$ denote the control inputs of switches. The lattice function evaluates to logic value 1 if there is a path between the top and bottom plates of the lattice. In a lattice with four-terminal switches, a path is a sequence of switches connected by taking horizontal and vertical moves. Thus, the lattice function can be written as the sum of products of control inputs of switches in each path from the top to bottom plates. Fig. 1(c) presents the 4×2 lattice function $f_{4\times2}$. The lattice function is unique and does not include any redundant products. For example, the path $x_1x_3x_4x_6x_5x_7$ in the 4×2 lattice is eliminated by $x_1x_3x_5x_7$.

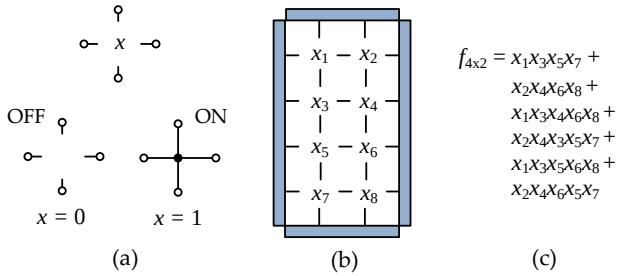


Fig. 1. (a) Four-terminal switch; (b) the 4×2 four-terminal switching network; (c) the 4×2 switching lattice function.

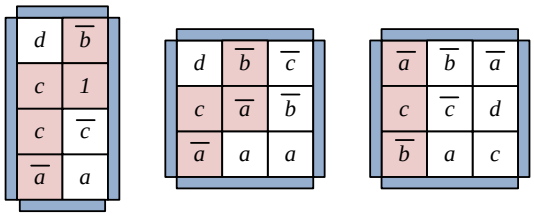
A switching lattice can be used to realize a logic function by simply finding an appropriate assignment to the control inputs of switches from the logic function variables and also, constant logic values 0 and 1. Thus, the fundamental problem, called lattice mapping (LM), is defined as: given a target function f and an $m \times n$ lattice, find an appropriate assignment to the control inputs of the given lattice such that f can be realized using the given lattice or prove that there exists no such assignment. Note that the LM problem is NP-complete [6]. The design complexity of a switching lattice is determined as the number of switches, *i.e.*, lattice size. Thus, the main optimization problem, called lattice synthesis (LS), is defined as: given a target function f, find an $m \times n$ lattice such that there exists an appropriate assignment to the control inputs of the given lattice and m times n is minimum. Over the years, exact and approximate algorithms have been introduced for the LS problem [6]–[13]. Recently, it is shown that switching lattices can be implemented under the CMOS technology [14]. It is observed that logic functions designed by switching lattices occupy significantly less area due to the dense and compact structure of a lattice and have comparable delay and power dissipation with respect to the conventional designs including two-terminal switches.

As an example, consider the target function $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$. Fig. 2(a) presents the realization of f using a switching lattice[1] with a minimum size found using the algorithm of [6]. However, this solution does not guarantee

L. Aksoy and M. Altun are with the Emerging Circuits and Computation Group, Department of Electronics and Communication Engineering, Istanbul Technical University, Maslak, 34469, Istanbul, Turkey (e-mail: {aksoyl, altunmus}@itu.edu.tr).

N. Akkan and H. Sedef are with the Department of Electronics and Communication Engineering, Yıldız Technical University, Esenler, 34220, Istanbul, Turkey (e-mail: {nakkan, sedef}@yildiz.edu.tr).

[1]Keeping the same order in the products and variables of $f_{4\times2}$ in Fig. 1(c), the function realized by the lattice can be given as $g = dcc\overline{a} + \overline{b}1\overline{c}a + dc1\overline{c}a + \overline{b}1cc\overline{a} + dcc\overline{c}a + \overline{b}1\overline{c}c\overline{a}$. After the application of Boolean algebra laws, it can be written as $g = \overline{a}cd + a\overline{b}\overline{c} + \overline{a}\overline{b}c$.



(a) (b) (c)

Fig. 2. Realizations of $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$ using switching lattices where a critical path is highlighted: (a) with a minimum lattice size; (b) under a delay constraint of 4; (c) under a minimum delay constraint of 3.

that the lattice realization has the minimum delay. To the best of our knowledge, previously proposed algorithms did not consider the delay of the design while searching for a solution with a minimum lattice size. In this article, we show that the delay of a switching lattice depends heavily on the longest path between the top and bottom plates, called *critical path*. It corresponds to an irredundant product in the lattice function and consists of a maximum number of switches. It does not include any constant logic value 0 or both a variable and its complement as a control input of a switch, meaning that there is an assignment to the control inputs of switches in the critical path which opens a way in between the top and bottom plates. The solution of [6] in Fig. 2(a) has a critical path with 5 switches corresponding to the $\overline{b}1cc\overline{a}$ product.

In this article, we initially present the CMOS-compatible four-terminal switch model of [14] and then, introduce the Elmore delay formulation of a path in a lattice using this four-terminal switch model and discuss the factors that have an impact on the delay of a lattice. Taking into account these factors, the Elmore delay estimation clearly indicates that the number of switches in the critical path has a significant effect on the worst-case delay of a lattice realization of a logic function. Inspired by this motivation, we introduce optimization algorithms, called PHAEDRA and TROADES, developed to find a realization of a logic function using a switching lattice with the smallest size while respecting the delay constraint dc given as the number of switches in the critical path. In PHAEDRA, the initial lower and upper bounds of the search space are determined respecting dc. Then, the search space is explored in a binary search manner where for each lattice candidate, the problem of checking if the target function can be realized using the lattice candidate respecting dc is formulated as a satisfiability (SAT) problem. The SAT problem includes the constraints to ensure that the target function is realized using the lattice candidate and as well as the constraints to avoid the violation of dc. Returning to our example, Figs. 2(b)-(c) present solutions of PHAEDRA with 3×3 lattices[2] when dc is 4 and 3, respectively, increasing the lattice size by only 1 with respect to the exact solution given in Fig. 2(a). In these figures, the switches in the critical path correspond to the $\overline{b}\overline{a}c\overline{a}$ and $\overline{a}c\overline{b}$ products, respectively. We note that in our simulations described in Section III, the delay of lattice realizations given in Figs. 2(a)-(c) is respectively found as $1.53ns$, $1.25ns$, and $1.02ns$ where a solution with a minimum dc leads to a 33% reduction in delay when compared to the solution with a minimum number of switches.

Experimental results show that PHAEDRA can reduce the number of switches in the critical path significantly, increasing the number of switches slightly when compared to the previously proposed algorithms. However, due to a dramatic increase in the SAT problem complexity as the number of variables and products increases in the target and lattice functions, it cannot handle logic functions with a large number of variables and products. Hence, in this article, we also introduce TROADES that partitions a large logic function into a number of small sub-functions, find their realizations in a binary search manner as done in PHAEDRA, and combine these solutions into a single lattice. Experimental results show that the solutions of TROADES on small size instances are close to those of PHAEDRA and its solutions on large size instances are obtained using less computational effort than PHAEDRA. Experimental results on the simulation of lattice realizations of logic functions clearly indicate that shortening the critical path can reduce the delay of a lattice realization significantly.

The rest of this article is organized as follows: Section II presents the background concepts and Section III introduces the Elmore delay formulation of a path in a switching lattice. Section IV describes the SAT encoding of the LM problem under a delay constraint and Section V introduces the proposed algorithms. Experimental results are given in Section VI and finally, Section VII concludes the article.

II. Background

A. Boolean Logic Function

A *logic function*, $f : \mathcal{B}^r \rightarrow \mathcal{B}$, over r variables $y_1, \ldots, y_r$ maps each truth assignment in $\mathcal{B}^r$ to 0 or 1. The logic function f in *sum of products* (SOP) form on r variables is a disjunction of s products $p_1, \ldots, p_s$, where a *product* $p_i = l_{i1} \cdot l_{i2} \cdot \ldots \cdot l_{ij}$, $i \leq s$ and $j \leq r$, is a conjunction of literals. A *literal* l_{ij}, $i \leq s$ and $j \leq r$, is either a variable y_j or its complement $\overline{y_j}$. A product is an *implicant* if and only if it evaluates f to 1 and it is a *prime implicant* if it is an implicant and there exists no other implicant whose literals are subset of its literals. In an *irredundant SOP* (ISOP) form of f, every product is a prime implicant and no product can be deleted without changing f. The *degree* of f, denoted as δ, is the maximum number of literals in the products of f.

A logic function φ in *product of sums* (POS) form on r variables is a conjunction of t clauses $c_1, \ldots, c_t$, where a *clause* $c_i = l_{i1} + l_{i2} + \ldots + l_{ij}$, $i \leq t$ and $j \leq r$, is a disjunction of literals. If a literal of a clause is set to 1, the clause is satisfied. If all literals of a clause are set to 0, the clause is unsatisfied. The *satisfiability* (SAT) problem is to find an assignment to the variables of a function φ in POS form that makes φ to be equal to 1 or to prove that φ is equal to 0. The SAT problem is NP-complete [15].

A *combinational circuit* is a directed acyclic graph with nodes and directed edges corresponding to logic gates and wires connecting the gates, respectively. The POS formula of

[2]The 3×3 lattice function can be given as $f_{3\times3} = x_1x_4x_7 + x_2x_5x_8 + x_3x_6x_9 + x_1x_4x_5x_8 + x_2x_5x_4x_7 + x_2x_5x_6x_9 + x_3x_6x_5x_8 + x_1x_4x_5x_6x_9 + x_3x_6x_5x_4x_7$. Starting from the top-left switch in the lattice with an index 1, indices of control inputs are determined as traversing each switch in the lattice to right and down and increasing the index by 1 as done in Fig. 1(b).

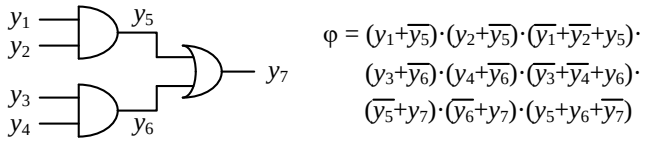


Fig. 3. A combinational network and its POS formula.

a combinational circuit is the conjunction of POS formula of each gate which denotes the valid input-output assignments to the gate. The derivation of POS formulas of basic logic gates can be found in [16], [17]. Fig. 3 shows a combinational circuit and its formula in the POS form.

B. Lattice Function

Recall that a path in a switching lattice is a sequence of switches connected by taking horizontal and vertical moves and a lattice function includes all irredundant paths between the top and bottom plates. Table I presents the number of products in an $m \times n$ lattice function at the top of each entry and degree of the lattice function at the bottom of each entry where $2 \leq m, n \leq 8$. The minimum number of variables in products of an $m \times n$ lattice function is equal to m. Thus, the number of switches in a path of an $m \times n$ lattice ranges between m and δ, *i.e.*, the degree of the lattice function $f_{m \times n}$.

Observe from Table I that the number of products and degree of lattice functions increase dramatically as the number of rows and columns increases, pointing out the lattices that can be used to realize a rich variety of logic functions. For the lattices with the same size, there exists a wide range of functions with different number of products and degrees. As an example, while $f_{3\times7}$ includes 49 products with a degree of 9, $f_{7\times3}$ has 163 products with a degree of 13. This is also true for the lattices with sizes very close to each other. For example, while $f_{6\times5}$ has 621 products with a degree of 16, $f_{4\times7}$ and $f_{8\times4}$ contain 203 and 1528 products with a degree of 12 and 17, respectively.

C. Elmore Delay

The Elmore delay model represents a circuit of interest as a resistance-capacitance (RC) tree network where the root of the tree is a voltage source and the leaves are the capacitors at the end of branches [18], as illustrated in Fig. 4. Note that the RC tree network does not include any resistor loops and thus, there is always a unique resistive path between each two nodes in the network. The delay is estimated from the switched source to one of the leaf nodes and computed as the sum over each node capacitance multiplied by the resistance on the shared paths from the source to the node and the leaf. Mathematically, the Elmore delay of a node i in an RC tree network is given as follows [19]:

$$\tau_i = \sum_{j=1}^{N} C_j \sum_{\substack{for\ all \\ k \in P_{ij}}} R_k$$

where N is the number of nodes and $P_{ij} = P_i \bigcap P_j$ indicates the common parts of the path from the input to the node i, *i.e.*, P_i and the path from the input to the node j, *i.e.*, P_j, where $i, j = 1, 2, \ldots, N$. As an example, τ_3 in the RC tree network

TABLE I
NUMBER OF PRODUCTS AND DEGREE OF AN $m \times n$ LATTICE FUNCTION.

m/n	2	3	4	5	6	7	8
2	2 2	3 2	4 2	5 2	6 2	7 2	8 2
3	4 4	9 5	16 6	25 7	36 8	49 9	64 10
4	6 5	17 6	36 7	67 10	118 11	203 12	344 13
5	10 7	37 9	94 11	205 13	436 16	957 18	2146 20
6	16 8	77 10	236 12	621 16	1668 19	4883 22	14880 24
7	26 10	163 13	602 16	1905 19	6562 23	26317 26	110838 29
8	42 11	343 14	1528 17	5835 22	25686 26	139231 30	797048 33

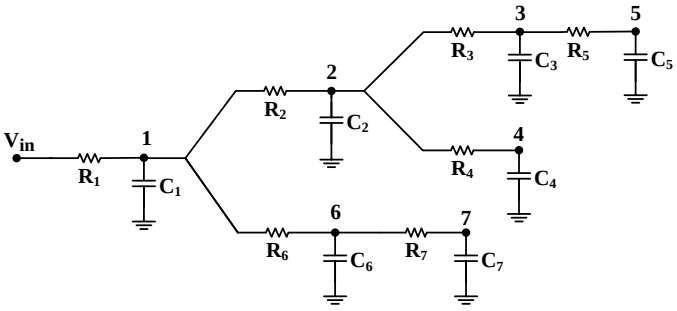


Fig. 4. An RC tree network with several branches.

of Fig. 4 is computed as $C_1R_1 + C_2(R_1 + R_2) + C_3(R_1 + R_2 + R_3) + C_4(R_1 + R_2) + C_5(R_1 + R_2 + R_3) + C_6R_1 + C_7R_1$.

Although the Elmore delay model has a limited accuracy and is restricted to be applied only for a step response delay, it is commonly used due to its simplicity [20].

D. Targeted Problems

The lattice mapping under a delay constraint (LM_DC) problem is defined as: given a target function f, an $m \times n$ lattice, and a delay constraint dc in terms of the number of switches in the critical path, find an appropriate assignment to the lattice variables realizing f and respecting dc or prove that there exists no such assignment. The lattice synthesis under a delay constraint (LS_DC) problem is defined as: given a target function f and a delay constraint dc, find an $m \times n$ lattice such that there exists an appropriate assignment to the lattice variables realizing f without violating dc and m times n is minimum. Note that the minimum value of dc is equal to δ, *i.e.*, the degree of the target function f.

E. Related Work

The exact method of [6] explores the search space of the LS problem in a dichotomic search manner in between the lower and upper bounds computed in [21]. The algorithm of [11] applies the same search strategy as the exact method, but also improves the upper bound of the search space in the LS problem and uses a SAT encoding for the LM problem. The methods of [7], [8], and [9] decompose a target function into smaller sub-functions by exploiting the p-circuits, D-reducible, and autosymmetric forms of the target function, respectively and merge the realizations of these sub-functions into a lattice. The method of [10] determines a number of promising lattice

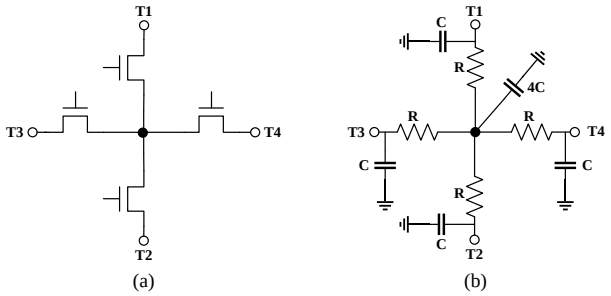


Fig. 5. (a) Model of the four-terminal switch including NMOS transistors; (b) RC tree network of the four-terminal switch model.

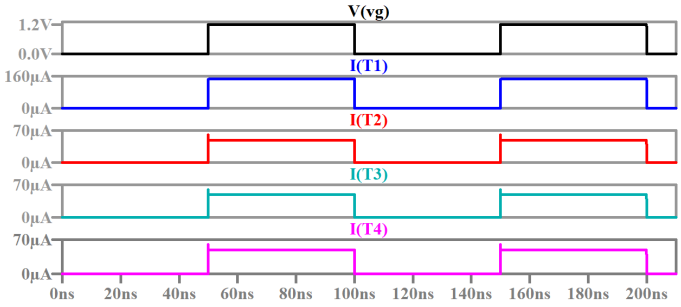


Fig. 6. Current waveforms in each terminal.

candidates and uses a method of [6] to find if one of these lattices leads to a solution. The divide and conquer method of [12] can easily handle logic functions with a large number of variables and products by partitioning such a logic function into smaller sub-functions, finding their realizations using the algorithm of [11], and merging these solutions into a single lattice. Moreover, the algorithm of [13] uses three techniques to decompose a complex logic function into sub-functions. To the best of our knowledge, there exists no algorithm proposed for the LM_DC and LS_DC problems.

III. Formulation of Delay in Switching Lattices

In this section, we introduce the Elmore delay formulation of a single path in a lattice formed based on the previously proposed four-terminal switch model of [14], leading to the estimation of the delay of a critical path which is the largest resistive path. We also discuss the factors that have a significant impact on the delay of a lattice.

In [14], it is shown that a four-terminal switch can be modeled using four equivalent NMOS transistors as illustrated in Fig. 5(a) and can be implemented using the CMOS technology. The model has actually six terminals and in this figure, two terminals, which are the control input, that is connected to the gate nodes of all transistors, and the body (bulk) terminal, that is grounded, are not shown for the sake of clarity. The model is symmetric and has six possible paths between its four terminals named as T1, T2, T3, and T4. The voltage levels of these terminals determine the current paths. As an example, assume that a pulse signal is applied to the control input when T1 is connected to the supply voltage and the other three terminals are grounded. In this case, there are three current paths and the current entering T1 is divided equally through other terminals since the model is symmetric. The current of each terminal is simulated in the

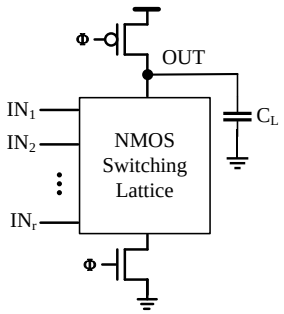


Fig. 7. Footed dynamic logic circuit including a switching lattice.

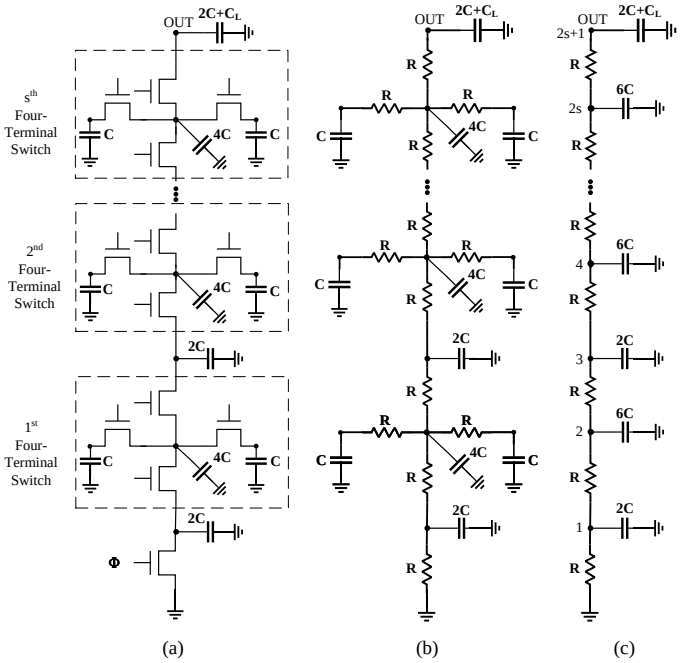


Fig. 8. (a) Footed dynamic logic circuit with cascaded s four-terminal switches; (b) corresponding RC tree network; (c) simplifed RC tree network of Fig. 8(b) by short-circuiting the off-path resistances with respect to the Elmore delay model.

LTspice environment and presented in Fig 6, indicating the proper behavior of the four-terminal switch. In this figure, vg denotes the control input of the four-terminal switch. In our simulations, all transistors have a minimum length and width of 60nm and 120nm according to the 65nm fabrication process, respectively. On the other hand, the RC tree network of the model is given in Fig. 5(b) where the nonlinear characteristics of a transistor are approximated using average resistance and capacitance over the switching range of the gate node for the delay estimation [18]. We define the average resistance value as R and the average capacitance value on drain or source node as C despite the nonlinear voltage dependence of the capacitors. At the interconnection of four transistors, the equivalent capacitance is taken as $4C$.

To simulate a switching lattice, it is used as a pull-down network of a footed dynamic logic circuit [14] shown in Fig. 7. The inputs to the switching lattice are the control inputs of four-terminal switches. The load capacitance C_L is connected to the output node that is precharged when the pull-up transistor, which is pulsed by a clock ϕ, is on. Thus, the output is evaluated when the pull-up transistor is off.

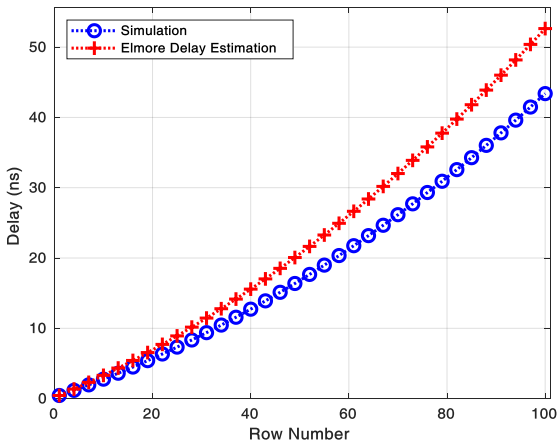


Fig. 9. Delay of $m \times 1$ switching lattices.

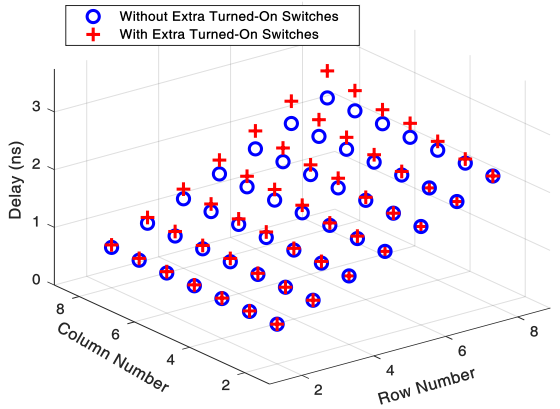


Fig. 10. Delay of $m \times n$ switching lattices including a single path with a minimum length.

The delay of a path including cascaded s four-terminal switches between the top and bottom plates of a switching lattice, can be estimated based on its RC tree network shown in Fig. 8. Analyzing this network, the delay of a path including s four-terminal switches is formulated as follows:

$$\tau_{OUT} = \sum_{\substack{i=1 \\ i \text{ odd}}}^{2s+1} i(R)(2C) + \sum_{\substack{i=2 \\ i \text{ even}}}^{2s} i(R)(6C) + (2s+1)RC_L$$

which can be simplified as

$$\tau_{OUT} = (8s^2 + 10s + 2)RC + (2s+1)RC_L.$$

Thus, the delay of a four-terminal switch can be estimated as $\tau_{OUT} = 20RC + 3RC_L$ when s is set to 1. Fig. 9 presents the simulated and estimated delay of an $m \times 1$ lattice where m ranges in between 1 and 100. Note that the estimated values are computed when C_L is $50fF$ and the average of effective resistance R and the capacitance C values are respectively found as $2.9k\Omega$ and $0.1fF$ in our simulations. Observe that the estimated delay values are close to the simulation results, being always higher than the simulation results.

The formulation of delay in a switching lattice becomes harder for the cases, where a lattice has a large number of rows and columns, includes extra turned-on four-terminal switches, which correspond to branches of the path, and has multiple isolated and unisolated paths. In following, we explore these factors, that have an impact on the delay of a switching lattice, in our simulations.

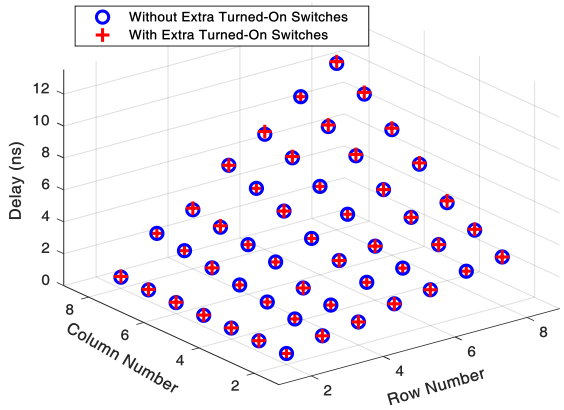


Fig. 11. Delay of $m \times n$ switching lattices including a single path with a maximum length.

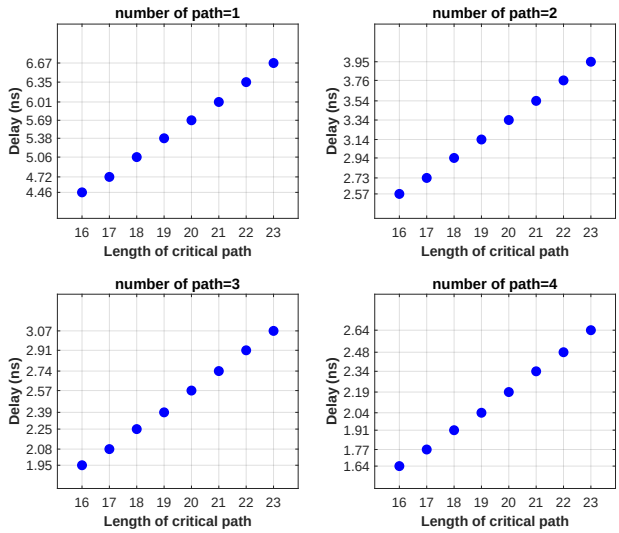


Fig. 12. Effect of the number of paths and number of switches in path on the delay of a 16×16 switching lattice.

Fig. 10 presents the delay of an $m \times n$ switching lattice in simulation, where $2 \leq m, n \leq 8$, when there is a single path which has m four-terminal switches. As mentioned in Section II-B, m is the minimum number of four-terminal switches in the critical path of an $m \times n$ lattice. In order to explore the delay affected by the four-terminal switches other than the ones in the path, in one scenario, all these switches are turned off and in another scenario, all of them are turned on, except the ones which can generate resistor loops. Observe that as the number of rows increases, the delay of a lattice increases. When all the four-terminal switches which are not in the path are turned off for lattices with m rows, the number of columns does not affect the delay. However, when there are extra turned-on switches other than the ones in the single path, ensuring that no resistor loop is generated, the delay of a lattice increases as the number of columns increases since the number of turned-on switches is increased in this case. Considering the Elmore delay model introduced in Section II-C, turning on the switches, that are not in the path, can create new branches on the RC tree and/or increase the length of existing branches. The capacitors depicted as leaves on branches of the RC tree contribute to the overall delay.

Similarly, Fig. 11 presents the delay of an $m \times n$ switching lattice in simulation, where $2 \leq m, n \leq 8$, when there is a

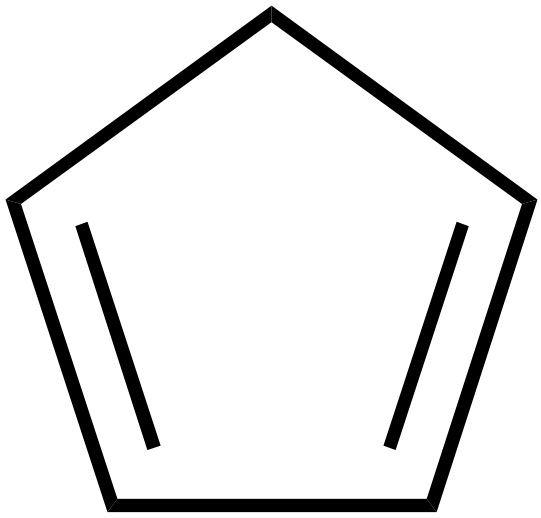


Fig. 13. Path examples: (a) a single; (b) two isolated; (c) two nonisolated.

single path which has a maximum number of four-terminal switches given as the degree of the lattice function in Table I. Observe that a critical path with a maximum number of switches dominates the delay of the lattice and the impact of extra turned-on switches is very little. This is simply because the number of extra turned-on switches is decreased when the single path includes a maximum number of switches.

In order to find the impact of multiple paths on the delay of a switching lattice, in a 16×16 lattice, we generated p paths which consist of s four-terminal switches, where p ranges between 1 and 4 and s is in between 16 and 23, and we turned off all the switches other than the ones in the paths. Note that the generated paths are isolated from each other, meaning that no four-terminal switch in a path is a neighbor of any four-terminal switch in another path. Fig. 12 presents the delay of 16×16 lattices in simulation under this scenario. Observe that a lattice with a single path as illustrated in Fig.13(a) has the largest delay with respect to those of lattices including multiple isolated paths as illustrated in Fig.13(b). This is because multiple paths are considered as parallel resistances whose equivalent resistance yields a smaller delay value when compared to the delay of lattice including a single path. Note that if multiple paths are unisolated as illustrated in Fig.13(c), there will be both serial and parallel switches in the paths and the delay will also be smaller than that of lattice including a single path. Recall that no resistor loop is permitted in the Elmore delay model and hence, the largest resistive path is taken into account to estimate the worst-case delay.

Although there exist other factors as shown above, the Elmore delay estimation of a lattice indicates that the number of switches in the critical path has a significant impact on the worst-case delay of a lattice. In the algorithms introduced in the next two sections, this metric, which is not complicated to achieve, is used as a delay constraint.

IV. SAT Encoding of the LM_DC Problem

In this section, we present the SAT formulation of the LM_DC problem which is the fundamental problem to be solved in the algorithms introduced in the next section.

Given the target function f, an $m \times n$ lattice, and a delay constraint dc, in order to check if f can be realized using the $m \times n$ lattice without violating dc, initially, the *structural check* procedure is applied. In this procedure, for each product of the target function with j literals, it is checked if there is a different product in the lattice candidate function with k literals, where $j \leq k \leq dc$. For our example in Fig. 2, $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$, assume that dc is equal to 3, which is the degree of f, and suppose that the 4×2 lattice is considered. Observe from Fig. 1(c) that all products of $f_{4\times2}$ have a number of literals greater than dc, failing the structural check procedure. On the other hand, when the 3×3 lattice is considered, there exists a different product of $f_{3\times3}$ that covers each product of f.

If the structural check is passed, the LM_DC problem is formulated as an SAT problem in two steps as given below. Note that the SAT encoding of the LM_DC problem is based on the one given for the LM problem in [11].

In the first step, the set LV, which includes the lattice function variables, and the set TL, which contains the target function literals and constants 0 and 1, are generated. The mapping variables $lv_i\_tl_j$, where $lv_i \in LV$, $1 \leq i \leq |LV|$, $tl_j \in TL$, $1 \leq j \leq |TL|$, and $|A|$ stands for the cardinality of set A, are generated. The mapping variable $lv_i\_tl_j$ indicates that the lattice variable lv_i is assigned to an element of TL, tl_j, when this mapping variable is set to high. For our example, $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$, suppose that the 3×3 lattice is considered when dc is 3. Thus, $LV = \{x_1, x_2, \ldots, x_9\}$, $TL = \{a, \overline{a}, \overline{b}, c, \overline{c}, d, 0, 1\}$, and for example, setting the mapping variable $x_7\_\overline{c}$ to 1 indicates the assignment of $\overline{c}$ of TL to x_7 of LV. To confirm that each lattice variable is assigned to only one element in TL, the necessary clauses are generated as follows:

$$\prod_{i=1}^{|LV|} \sum_{j=1}^{|TL|} lv_i\_tl_j \quad \text{and} \quad \prod_{i=1}^{|LV|} \prod_{j=1}^{|TL|-1} \prod_{k=j+1}^{|TL|} \overline{lv_i\_tl_j} + \overline{lv_i\_tl_k}$$

where $\prod$ and $\sum$ denote AND and OR operators, respectively. While the former clauses ensure that for each lattice variable, at least one of the mapping variables should be high, the latter ones guarantee that for each lattice variable, when one mapping variable is high, the others should be low.

In the second step, to satisfy the functionality of f, for each truth table entry, a combinational circuit, which corresponds to the lattice function, is generated and the value of f at this truth table entry is assigned to the circuit output. The circuit inputs, which are the lattice function variables, are associated with the truth table entry and are denoted as lv_{i_tte}, where $1 \leq i \leq |LV|$ and tte is the truth table entry. The POS formula of the combinational circuit is obtained as shown in Fig. 3 and is simplified based on the logic value at the output. For our example, $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$, considering the 3×3 lattice when dc is 3, Fig. 14 shows the circuits generated for $abcd = 0000$ and $abcd = 1000$ where f evaluates to logic 0

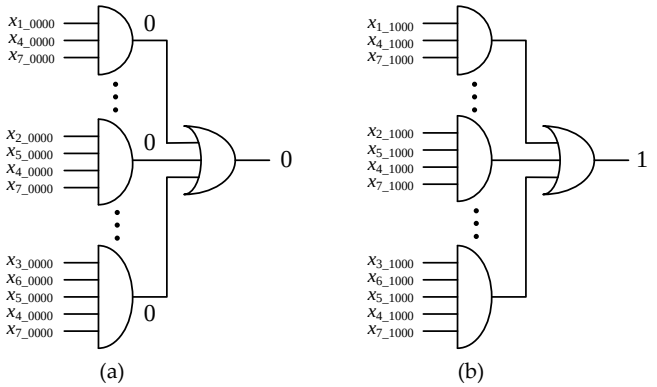


Fig. 14. The circuits of $f_{3\times3}$ for $f = \bar{a}\bar{b}c + a\bar{b}\bar{c} + \bar{a}cd$: (a) when $abcd = 0000$ and f is low; (a) when $abcd = 1000$ and f is high.

and 1 values, respectively. In this figure, only three products of $f_{3\times3}$ are shown for the sake of clarity.

In order to satisfy dc, for each truth table entry where f evaluates to logic value 1, we generate constraints to ensure that all the products of a lattice function, that include a number of literals greater than dc, evaluate to logic value 0. For our example, considering the 3×3 lattice when dc is 3 and $abcd = 1000$, the clauses, such as $(\overline{x_{2\_1000}} + \overline{x_{5\_1000}} + \overline{x_{4\_1000}} + \overline{x_{7\_1000}})$ and $(\overline{x_{3\_1000}} + \overline{x_{6\_1000}} + \overline{x_{5\_1000}} + \overline{x_{4\_1000}} + \overline{x_{7\_1000}})$, guarantee that in a path of lattice, which includes a number of switches greater than dc, at least one of its switches is off.

In order to link the mapping variables to the circuit inputs, for each mapping variable, we generate clauses which ensure that when it is set to high, the associated circuit input is set to a value determined by each truth table entry. For our example, when $abcd = 0000$, the constraints, such as $x_{1\_}a \Rightarrow \overline{x_{1\_0000}}$ and $x_{3\_}\bar{b} \Rightarrow x_{3\_0000}$, where $\Rightarrow$ denotes the implication operator, ensure that the circuit input has the corresponding value when a lattice variable is assigned to a target literal. When a lattice variable is assigned to a constant value 0 or 1, the related circuit input is set to that value.

Thus, we generate a SAT problem that formalizes the LM_DC problem and is solved using a SAT solver. For the SAT solver, we set a time limit as 1200 seconds, determined empirically. Thus, if the SAT solver finds a solution in the given time limit, the assignment to the lattice variables is obtained by the mapping variables set to high. Otherwise, it is accepted that the target function cannot be realized using the given lattice.

The performance of algorithms developed for the LS_DC problem described in the following section heavily depends on solving the SAT problems generated for the LM_DC problems. We note that as the number of variables and products in the target and lattice function increases, the SAT problem size may increase dramatically, going beyond the limitations of the state-of-art SAT solvers.

In order to analyze the increase in the complexity of SAT problems, we consider the logic function of an r-input XOR gate, denoted as r-XOR. Note that r-XOR includes all possible $2r$ literals and consists of $2^{(r-1)}$ products, each having r literals. Fig. 15 presents the number of variables and clauses (in the logarithmic scale) of the SAT problems generated for r-XOR on three possible lattices with the smallest size when dc is set to the degree of the target function, *i.e.*, r.

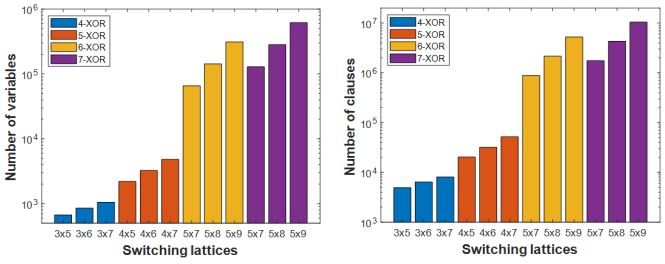


Fig. 15. Complexity of SAT problems on XOR functions: (a) number of variables; (b) number of clauses.

Observe from Fig. 15 that the complexity of the SAT problem increases dramatically as the number of inputs in the XOR logic function, and consequently, the number of products, increase. For example, the SAT problem, which is generated to check if 6-XOR (7-XOR) can be realized using the 5×9 lattice, includes 311,286 (618,960) variables and 5,235,810 (10,446,973) clauses. Moreover, the SAT problem complexity increases as the lattice size increases because the number of products and degree of the lattice function increase as shown in Table I. For example, for 7-XOR on the 5×7 (5×8) lattice, the SAT problem has 128,816 (281,856) variables and 1,751,515 (4,305,693) clauses. This analysis clearly indicates that the performance of LS_DC algorithms, which solve LM_DC problems formulated as SAT problems, depends heavily on the number of literals and products of the target function and lattice size.

V. Algorithms Proposed for the LS_DC Problem

In this section, we introduce the algorithms developed for the LS_DC problem, namely a binary search algorithm PHAEDRA and a divide and conquer method TROADES. These algorithms take the ISOP form of a logic function as a target function f and the delay constraint dc and return the realization of f using a switching lattice which respects dc.

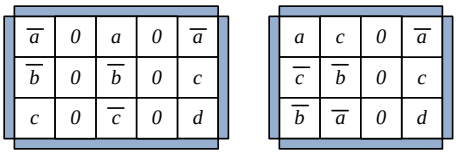
A. PHAEDRA: *A Binary Search Algorithm*

PHAEDRA is based on JANUS [11] proposed for the LS problem and its main steps are given as follows:

1) Compute the *lower bound* (lb) and *upper bound* (ub) of the LS_DC problem, both in terms of the number of switches.
2) If $lb = ub$, return found solution while computing ub.
3) Determine the *middle point* as $mp = \lfloor (lb + ub)/2 \rfloor$ and generate the set of lattice candidates C.
4) For each lattice candidate in C, formulate the related LM_DC problem as a SAT problem as described in Section IV and solve it using a SAT solver. If there exists a solution, set ub to mp and go to Step 6.
5) If there are no solutions for all lattice candidates in C, set lb to $mp + 1$.
6) If $lb < ub$, go to Step 3. Otherwise, return the solution.

In the first step of PHAEDRA, the initial lower bound of the LS_DC problem is computed as follows:

1) Set the initial lower bound lb to 1.



(a) (b)

Fig. 16. Realizations of $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$ when dc is 3: (a) PS method [6]; (b) THYESTES.

2) Determine all lattices with size lb. Apply the structural check procedure described in Section IV to each lattice. If the structural check procedure is passed, return lb.
3) If the structural check procedure is failed on all lattices with the size lb, increase lb by 1 and go to Step 2.

For our example in Fig. 2, $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$, suppose that lb and dc are equal to 8 and 3, respectively. There are 4 lattices with the size of 8, *i.e.*, the 1×8 lattice with $f_{1\times8} = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8$, the 2×4 lattice with $f_{2\times4} = x_1x_5 + x_2x_6 + x_3x_7 + x_4x_8$, the 4×2 lattice with $f_{4\times2}$ as shown in Fig. 1(c), and the 8×1 lattice with $f_{8\times1} = x_1x_2x_3x_4x_5x_6x_7x_8$. Observe that while all products of $f_{1\times8}$ and $f_{2\times4}$ have a number of literals less than the degree of f, all products of $f_{4\times2}$ and $f_{8\times1}$ have a number of literals greater than dc. For our example, the initial lower bound is computed as 9 where the 3×3 lattice passes the structural check.

In the first step of PHAEDRA, to find the initial upper bound of the LS_DC problem, we apply two methods, namely THYESTES given below and TROADES described in the following subsection, and determine ub as the minimum number of switches in the solutions of these techniques. THYESTES is based on the product separation (PS) method of [6] which places all s products of a target function on the columns of a lattice, separating each of them by an isolation column full of logic value 0s and filling the unspecified entries of the lattice with logic value 1. The PS method finds a solution with a $\delta \times (2s-1)$ lattice, where δ is the degree of the target function. Since its solutions have δ rows and the minimum value of dc is δ, dc is never violated. For our example, $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$, when dc is 3, its solution is shown in Fig. 16(a).

The PS method can be improved by reducing the number of isolation columns. In THYESTES, for each product p_i, we check if there exists another product p_j, such that both of them can be realized in a $\delta \times 2$ lattice respecting dc. Such a solution can be found by encoding the related LM_DC problem as a SAT problem as described in Section IV. If there exists such a solution, it is placed on a lattice. If there exists no product that can be realized with p_i, the product p_i is placed on a column of a lattice. If the product p_i is not the last product in the target function, an isolation column is inserted into the lattice. Note that since the LM_DC problems to be solved include lattices with only 2 columns, they can be solved easily. For our example, $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$, when dc is 3, the solution of THYESTES is shown in Fig. 16(b).

In PHAEDRA, the lattice candidates are explored in between the lower and upper bounds. In its third step, the set of lattice candidates C is generated as follows:

$$C = \begin{cases} (m,n) \mid & m \times n \leq mp \\ & \forall (m',n') \notin F, m' = m \text{ and } n' \geq n \end{cases}$$

where F is the set that includes the row and column of lattices on which it is proved that the given target function cannot be realized without violating dc as described in Section IV.

Observe that as the difference between the initial upper and lower bounds of the LS_DC problem increases, the number of lattice candidates, and consequently, the number of LM_DC problems, increase. Hence, it is important to find a good initial upper bound using a little computational effort. The following subsection presents TROADES used to find an initial upper bound for the LS_DC problem.

B. TROADES: *A Divide and Conquer Method*

TROADES is based on MEDEA [12] proposed for the LS problem and has four main steps described as follows:

1) Compute the initial lower bound of the LS_DC problem, lb, and its initial upper bound using THYESTES, ub, as described in Section V-A.
2) If the difference between the initial upper and lower bounds, $dulb$, is greater than 31, partition the target function into smaller sub-functions until the $dulb$ value of all sub-functions is less than or equal to 31. Otherwise, consider the target function as a sub-function.
3) Find the realization of each sub-function respecting dc in a binary search manner as done in PHAEDRA.
4) If there exist multiple sub-functions, combine these realizations into a single lattice and explore alternative implementations of sub-functions.

After computing the lower and upper bounds of the search space in the first step, if the $dulb$ value for the target function is greater than 31, in an iterative manner, products of the target function are partitioned into two sub-functions. As a simple example, consider the iterative partitioning of f as $f = g + h$, $g = f_1 + f_2$, $h = f_3 + f_4$, where $f = f_1 + f_2 + f_3 + f_4$. Note that while the functions f, g, and h have a $dulb$ value greater than 31, the functions f_1, f_2, f_3, and f_4 have a $dulb$ value less than or equal to 31. Note that the products of a function are partitioned into two sub-functions such that the number of products in a sub-function is equal to or is one less than that of the other and the products in a sub-function share a maximum number of variables. While determining the $dulb$ value, two criteria were considered. On one hand, it is desired to find the solutions of sub-functions using a little computational effort with PHAEDRA. On the other, the realizations of these sub-functions are desired to yield a solution with a small lattice size. Although these two criteria conflict with each other, it was found that the $dulb$ value 31 meets them adequately.

In the third step, the solutions of all k sub-functions are found in a binary search manner using PHAEDRA where only THYESTES is used to find the initial upper bound of the search space. After the solution of each sub-function f_i, denoted as $m_i \times n_i$, where $1 \leq i \leq k$, is found, the maximum row of these solutions, mr, is determined as $max_{i=1}^{k}(m_i)$. Then, for each sub-function f_i, where $1 \leq i \leq k$, if $m_i \neq mr$, we

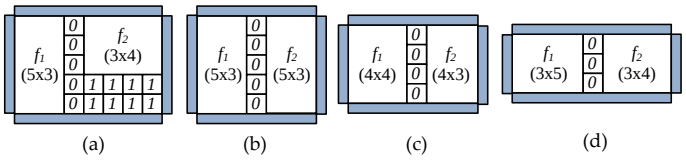


Fig. 17. (a) Merging the realizations of sub-functions into a single lattice; Realizations of sub-functions leading to different sizes respecting dc: (b) 5×7 lattice; (c) 4×8 lattice; (d) 3×10 lattice;

find its solution respecting dc on an $mr \times c$ lattice, where c is initially set to $\lceil (m_i n_i)/mr \rceil$ and is increased by 1 until a solution is found. By doing so, all sub-functions are realized using switching lattices with the same number of rows and thus, it is ensured that dc is not violated when these sub-functions are merged into a single lattice.

If there exist multiple sub-functions, in the beginning of the forth step, we keep the solution of each sub-function found in the previous step as $m'_i \times n'_i$, where $1 \leq i \leq k$. Then, we merge the solutions of sub-functions into a single lattice, called best lattice bl with the best lattice size bls set to $mr(k-1+\sum_{i=1}^{k} n_i)$. If $mr > 3$, we check if each sub-function f_i, where $1 \leq i \leq k$, can be realized using an $(mr-1)\times c$ lattice without violating dc, where c is initially set to $\lceil (m'_i n'_i)/(mr-1) \rceil$ and is increased by 1 until a solution is found. Whenever all the solutions of sub-functions are found with a row of $mr-1$, they are merged into a single lattice sl with its size sls. If sls is less than bls, then bl and bls are updated and this procedure is iterated. Otherwise, it is terminated.

As a simple example, suppose that a target function is partitioned into two sub-functions, f_1 and f_2, and PHAEDRA respectively finds a solution with 5×3 and 3×4 lattices without violating dc. A single lattice combining these two sub-functions is depicted in Fig. 17(a). Observe that this single lattice may not respect dc, even though the realization of each sub-function does not violate dc. This is simply because of logic 1s added under the realization of f_2. Hence, it is checked if f_2 can be realized by a lattice with a row of 5. Assume that it can be realized using the 5×3 lattice respecting dc as shown in Fig. 17(b). Thus, the single lattice includes 35 switches. However, if f_1 and f_2 can be realized using 4×4 and 4×3 lattices, respectively, the single lattice has 32 switches as shown in Fig. 17(c). Finally, if f_1 and f_2 can be realized using 3×5 and 3×4 lattices, the complexity of the single lattice reduces to 30 as shown in Fig 17(d).

When the $dulb$ value of the target function is less than or equal to 31, no sub-function is generated and the realization of the target function on a switching lattice is found in a binary search manner as done in PHAEDRA. For our example in Fig. 2, $f = \overline{a}\overline{b}c + a\overline{b}\overline{c} + \overline{a}cd$, when dc is 3, the initial lower and upper bounds are computed as 9 and 12, respectively and thus, the solution of TROADES is the same as the solution of PHAEDRA given in Fig. 2(c).

VI. Experimental Results

In this section, we present the results of PHAEDRA and TROADES proposed for the LS_DC problem and the algorithms of [6], [7], [10]–[12] introduced for the LS problem. Note that the proposed algorithms developed in Perl use *espresso* [22] to find the ISOP forms of target functions and *glucose4.1* [23] to solve a SAT problem. They can be found at https://github.com/leventaksoy/Lattices. We used the updated version of the exact method of [6] where an issue, that may cause the method to miss some paths in a switching lattice, was fixed [10]. The performance of algorithms [6], [10] was increased when the time limit for the SAT solver was set to 1200 seconds as done in the algorithm of [11]. The time limit for the SAT solver in the algorithm of [7] was 600 seconds as mentioned in [7]. All the algorithms were run on an Intel Xeon CPU at 2.40GHz with 28 cores and 128GB RAM with the time limit of 6 hours. We note that the time-out parameter in the SAT algorithm is based on the maximum run-time in seconds because i) the run-time of an LS algorithm is an important parameter while assessing the algorithms; ii) different lattices lead to SAT problems with different number of variables and clauses for the same target function, complicating the decision on the constant value of another time-out parameter. However, it is important to state that the given results of algorithms may not be reproducible due to a different computing environment.

TABLE II
DETAILS OF LOGIC FUNCTIONS.

Instance	in	pi	δ
5xp1_3	6	14	5
b12_06	9	9	6
dc1_02	4	4	3
ex5_23	8	12	4
ex5_27	8	11	4
misex1_02	7	5	5
misex1_06	6	5	4
mp2d_02	11	10	4
apex4_015	9	13	8
apex4_017	9	12	8
apex4_018	9	14	8
clip_00	9	21	6
clip_04	9	20	6
sao2_01	10	20	10
sao2_02	10	22	4
sao2_03	10	21	5

As an experiment set, we used 16 instances of [24]. Table II presents the details of these functions where in, pi, and δ stand for the number of inputs and prime implicants and degree of the logic function, respectively. Note that while the product of these three parameters is less than or equal to 486 for the first half of instances, this product is greater than or equal to 864 for the second half of instances.

To demonstrate the complexity of SAT problems generated for the LM_DC problem with different lattices when dc is equal to δ and $\delta + 1$, we used two logic functions, namely 5xp1_1 and mp2d_02 of Table II. Tables III and IV present the details on the SAT problems where *variables* and *clauses* stand for the number of variables and clauses, respectively. In these tables, *decision* denotes the result of the SAT solver, where *sat*, *unsat*, and *undet* indicate that it is respectively proved, disproved, and undetermined that the target function can be realized using the given lattice. Finally, *CPU* denotes the run-time of the SAT solver.

Observe from Tables III and IV that as the lattice size increases, the complexity of the SAT problem also increases, going beyond the capabilities of a state-of-art SAT solver [23]. Note also that the complexity of the SAT problem increases,

TABLE III
SUMMARY OF DETAILS ON THE SAT PROBLEMS GENERATED FOR THE 5XP1_3 INSTANCE.

Lattice	$dc = \delta$				$dc = \delta + 1$			
	variables	clauses	decision	CPU	variables	clauses	decision	CPU
4x8	14,272	175,848	undet	1200.0	14,272	173,544	undet	1200.0
4x9	22,072	296,479	undet	1200.0	22,072	293,791	undet	1200.0
4x10	34,480	504,294	sat	682.8	34,480	501,222	undet	1200.0
4x11	54,312	862,101	sat	1095.2	54,312	858,645	sat	805.3
5x6	17,060	220,365	unsat	6.7	17,060	218,445	undet	1200.0
5x7	34,250	500,610	unsat	24.0	34,250	498,306	sat	791.3
5x8	72,816	1,199,635	unsat	163.3	72,816	1,196,947	undet	1200.0
5x9	157,398	2,865,538	undet	1200.0	157,398	2,862,466	undet	1200.0

TABLE IV
SUMMARY OF DETAILS ON THE SAT PROBLEMS GENERATED FOR THE MP2D_02 INSTANCE.

Lattice	$dc = \delta$				$dc = \delta + 1$			
	variables	clauses	decision	CPU	variables	clauses	decision	CPU
3x10	306,120	2,755,403	undet	1200.0	306,120	2,722,635	undet	1200.0
3x11	359,117	3,298,228	undet	1200.0	359,117	3,261,364	undet	1200.0
3x12	416,184	3,906,227	sat	494.9	416,184	3,865,267	sat	183.8
3x13	477,321	4,583,470	sat	423.1	477,321	4,538,414	sat	364.4
4x7	513,632	5,187,590	unsat	46.2	513,632	5,138,438	unsat	772.2
4x8	814,928	8,941,170	unsat	117.4	814,928	8,883,826	undet	1200.0
4x9	1,299,374	15,562,460	unsat	212.7	1,299,374	15,496,924	undet	1200.0
4x10	2,076,860	27,136,758	unsat	255.8	2,076,860	27,063,030	undet	1200.0

as the number of inputs of the target function, *i.e.*, 6 for the 5xp1_3 function and 11 for the mp2d_02 function, increases. While the SAT problems generated for different dc values have the same number of variables, the ones generated for a higher dc value have a smaller number of clauses. This is simply because as dc increases, the number of critical paths, whose length is greater than dc, decreases. Note also that the number of SAT problems, which are generated under a small dc value and solved in the given time limit, is greater than that of SAT problems generated under a large dc value. The reason for that a small dc value constrains the search space more than a large dc value. For this reason, it is interesting to observe that the SAT problems with a large number of variables and clauses may not require a large computational effort, *e.g.*, the ones obtained on lattices 4×11 and 5×8 for the 5xp1_3 function and the ones obtained on lattices 3×13 and 4×10 for the mp2d_02 function when dc is equal to δ.

Tables V and VI show the results of algorithms developed for the LS and LS_DC problems, respectively. In these tables, *sol*, *cp*, and *CPU* denote the solution of algorithms, number of switches in the critical path, and run-time of algorithms in seconds, respectively. Note that the *cp* values given in italic are inexact and could be found under a time limit of 300 seconds. Also, in Table VI, *sf* stands for the number of generated sub-functions in TROADES. The results of proposed algorithms were obtained when dc was set to δ and $\delta + 1$.

Observe from Tables V and VI that the algorithms proposed for the LS problem obtain solutions with a large number of switches in the critical path when compared to the degree of a logic function. The average number of switches in the critical path in the solutions of algorithms [7], [10], [6], [12] [11] are 2.4×, 3×, 1.3×, 1.5×, and 2× larger than the one in the solutions of PHAEDRA and TROADES when dc is equal to δ, respectively. Moreover, the solutions of PHAEDRA and TROADES include less number switches than those of algorithms developed for the LS problem on average, except MEDEA and JANUS. This is due to the fact that the algorithms of [7], [10], and [6] cannot handle the hard instances efficiently. Furthermore, as dc is increased, the number of switches in the solutions of PHAEDRA and TROADES decreases on average. Note that while the solutions of PHAEDRA are close to those of JANUS on moderate instances, its solutions are getting far away from those of JANUS on hard instances. One reason for PHAEDRA obtaining worse solutions than JANUS on hard instances is that a solution respecting dc requires a lattice with a large size, increasing the SAT problem complexity dramatically. Another reason is that the initial upper bound of the search space in the LS_DC problem obtained by PHAEDRA is larger than that of the search space in the LS problem found by JANUS. On the other hand, TROADES can find solutions on all those instances using the least computational effort on average. Note that the solutions of PHAEDRA on hard instances are exactly the same as the solutions of TROADES which were actually found as an initial upper bound in PHAEDRA. In these cases, PHAEDRA could not improve the initial upper bound while exploring the search space in a binary search manner.

Table VII presents the impact of the $dulb$ value on the solution quality and performance of TROADES. For this experiment, the $dulb$ value was set to 11 and 51. Recall that the results of TROADES in Table VI were found when $dulb$ is 31.

Observe from Tables VI and VII that as the $dulb$ value increases, the number of switches in the solutions of TROADES decreases, but increasing the run-time. This is mainly because the number of generated sub-functions is decreasing, leading to sub-functions with a large number of products. We note that when dc is equal to δ, setting the $dulb$ value to 51 reduces the number of switches on average 6.6% and 21.5% compared to the solutions obtained when $dulb$ value is 31 and 11, respectively. In this case, the run-time of TROADES is increased by 8.7× and 10.3× when compared to those when $dulb$ value is 31 and 11, respectively.

To simulate the lattice realizations, we generated LTspice netlists of the solutions of JANUS with a minimum number of switches on average and of the solutions of PHAEDRA

TABLE V
SUMMARY OF RESULTS OF ALGORITHMS PROPOSED FOR THE LS PROBLEM.

Instance	[7]			[10]			Exact [6]			MEDEA [12]			JANUS [11]		
	sol	cp	CPU	sol	cp	CPU	sol	cp	CPU	sol	cp	CPU	sol	cp	CPU
5xp1_3	4x11	7	11.1	15x4	26	1750.1	4x8	8	11698.6	5x8	9	55.8	4x9	15	19745.8
b12_06	5x10	9	22.6	5x4	9	24.4	5x4	9	144.7	5x4	10	23.8	5x4	10	23.8
dc1_02	3x5	5	0.1	3x4	5	0.2	4x3	6	0.2	4x3	5	0.2	4x3	5	0.3
ex5_23	4x11	9	13.2	4x8	12	1241.2	3x9	7	3810.3	3x12	8	29.9	3x9	9	3726.4
ex5_27	4x11	8	7.8	4x6	7	52.4	4x6	8	1436.1	3x9	6	1.7	3x8	7	1229.3
misex1_02	5x5	7	1.4	5x4	8	37.7	5x4	8	52.2	5x4	8	23.7	5x4	8	19.7
misex1_06	5x6	7	0.8	3x5	4	2.2	5x3	7	5.0	5x3	7	1.7	5x3	7	1.3
mp2d_02	4x13	11	0.4	4x9	14	46.1	4x7	12	4022.8	3x10	11	0.3	4x7	12	948.9
apex4_015	4x19	12	8520.1	31x6	*31*	21600.0	8x25	8	21600.0	5x11	9	16.7	5x10	12	21600.0
apex4_017	4x22	13	7419.7	8x23	8	21600.0	8x23	8	21600.0	5x15	9	14.2	7x7	11	21600.0
apex4_018	22x14	*38*	21600.0	43x5	*43*	21600.0	8x27	8	21600.0	6x13	10	859.4	7x8	11	21600.0
clip_00	5x14	11	341.5	6x41	6	21600.0	6x41	6	21600.0	5x16	11	26.5	5x11	13	21600.0
clip_04	5x10	12	146.7	6x39	6	21600.0	6x39	6	21600.0	4x13	7	14.9	4x11	11	21600.0
sao2_01	25x17	*36*	21600.0	10x39	10	21600.0	10x39	10	21600.0	8x18	12	1979.0	12x7	18	21600.0
sao2_02	5x15	12	266.8	23x7	*42*	21600.0	4x43	4	21600.0	4x18	7	11.3	4x13	12	21600.0
sao2_03	5x14	15	3495.2	21x8	*38*	21600.0	5x41	5	21600.0	5x18	10	200.0	6x13	19	21600.0
Avg. (1-8)	38.0	7.9	7.2	27.4	10.6	394.3	22.3	8.1	2646.2	25.0	8.0	17.1	22.8	9.1	3211.9
Avg. (9-16)	145.3	18.6	7923.8	223.0	23.0	21600.0	230.9	6.9	21600.0	79.4	9.3	417.4	58.5	13.4	21600.0
Avg. (1-16)	91.6	13.3	3965.5	125.2	16.8	10997.1	126.6	7.5	12123.1	52.9	8.7	203.7	40.6	11.3	12406.0

TABLE VI
SUMMARY OF RESULTS OF ALGORITHMS PROPOSED FOR THE LS_DC PROBLEM.

Instance	TROADES - $dc = \delta$				TROADES - $dc = \delta + 1$				PHAEDRA - $dc = \delta$			PHAEDRA - $dc = \delta + 1$		
	sol	cp	sf	CPU	sol	cp	sf	CPU	sol	cp	CPU	sol	cp	CPU
5xp1_3	4x13	5	3	12.4	4x12	6	3	7.0	4x10	5	7027.1	5x7	6	10799.6
b12_06	5x7	6	2	9.1	5x7	7	2	16.1	5x6	6	2153.1	4x6	7	1686.9
dc1_02	3x4	3	1	0.3	3x4	4	1	0.3	3x4	3	0.4	3x4	4	0.4
ex5_23	3x13	4	2	3.4	3x12	5	2	11.6	3x9	4	491.3	3x9	5	2896.0
ex5_27	3x12	4	2	15.8	3x11	5	2	11.9	3x8	4	1343.9	3x8	5	1463.0
misex1_02	4x5	5	1	8.1	4x5	6	1	23.4	4x5	5	8.4	4x5	6	23.4
misex1_06	3x5	4	1	0.7	3x5	4	1	2.0	3x5	4	0.8	3x5	4	2.1
mp2d_02	3x13	4	2	3.4	3x12	5	1	4975.8	3x12	4	2922.6	3x12	5	4991.1
apex4_015	5x15	8	4	13.1	5x15	9	4	10.9	5x15	8	21600.0	5x15	9	21600.0
apex4_017	4x21	8	4	108.0	4x21	8	4	30.0	4x21	8	21600.0	4x21	8	21600.0
apex4_018	4x21	8	4	32.4	4x21	9	4	25.9	4x21	8	21600.0	4x21	9	21600.0
clip_00	4x21	6	5	27.7	4x21	7	5	36.0	4x21	6	12814.3	4x21	7	21600.0
clip_04	4x18	6	5	18.1	4x15	7	4	12.2	4x18	6	15268.2	4x15	7	21600.0
sao2_01	7x27	10	8	384.9	7x27	11	8	229.7	7x27	10	21600.0	7x27	11	21600.0
sao2_02	3x34	4	5	21.4	4x19	5	4	34.9	3x34	4	21600.0	4x19	5	21600.0
sao2_03	4x28	5	5	69.6	4x23	6	4	1674.8	4x28	5	21600.0	4x23	6	21600.0
Avg. (1-8)	31.0	4.4	1.8	6.6	29.4	5.3	1.6	631.0	25.5	4.4	1743.4	24.1	5.3	2732.8
Avg. (9-16)	100.3	6.9	5.0	84.4	93.0	7.8	4.6	256.8	100.3	6.9	19710.3	93.0	7.8	21600.0
Avg. (1-16)	65.6	5.6	3.4	45.5	61.2	6.5	3.1	443.9	62.9	5.6	10726.9	58.6	6.5	12166.4

under the footed dynamic logic circuit using the four-terminal switch model described in Section III. In these netlists, the load capacitance was set to $50fF$. We used 10000 random inputs to simulate the behavior of the lattice realizations when the clock frequency was $10MHz$. Table VIII presents the delay in ns and average power dissipation in μW of these realizations.

Observe from Table VIII that the realization of logic functions on switching lattices with a minimum number of switches in the critical path can reduce the delay of the design significantly. In this case, the solutions of PHAEDRA lead to a 22% reduction on the delay of a lattice realization on average where the maximum gain is obtained as 59% on the 5xp1_3 function. However, as observed in clip_04, a solution with a minimum length of critical path does not always lead to the smallest delay due to the other factors described in Section III. In this case, the number of turned-on extra switches in the solution of PHAEDRA is larger than that in the solution of JANUS. Moreover, when dc is increased, the delay of lattice realizations is increased on average, although there are cases where the delay of a lattice realization with a large number of switches in the critical path is smaller than that of lattice realization including a small number of switches in the critical path, such as misex1_02, misex1_06, and apex4_018. On the other hand, the solutions of PHAEDRA lead to lattice realizations with power dissipation slightly larger than those of designs obtained by JANUS on average because of a large number of switches in the solutions of PHAEDRA.

VII. CONCLUSIONS

This article introduced the delay formulation of a path in a switching lattice and addressed the problem of realizing a logic function using a switching lattice with the fewest number of four-terminal switches under a delay constraint defined in terms of the number of switches in the critical path. The problem of checking if the logic function can be realized using the given lattice without violating the delay constraint was formulated as a SAT problem. In this article, a dichotomic search algorithm PHAEDRA and a divide and conquer method TROADES were introduced. Experimental results indicated that PHAEDRA can reduce the number of switches in a critical path

TABLE VII
SUMMARY OF RESULTS OF TROADES WITH DIFFERENT $dulb$ VALUES.

Instance	$dulb = 11$ - $dc = \delta$				$dulb = 11$ - $dc = \delta + 1$				$dulb = 51$ - $dc = \delta$				$dulb = 51$ - $dc = \delta + 1$			
	sol	cp	sf	CPU	sol	cp	sf	CPU	sol	cp	sf	CPU	sol	cp	sf	CPU
5xp1_3	4x17	5	5	2.6	4x17	6	5	2.6	4x12	5	2	33.1	4x10	6	2	25.3
b12_06	5x12	6	4	11.5	5x12	7	4	2.8	5x7	6	2	10.7	5x7	7	2	17.6
dc1_02	3x4	3	1	0.3	3x4	4	1	0.3	3x4	3	1	0.3	3x4	4	1	0.3
ex5_23	3x16	4	4	1.5	3x16	5	4	1.5	3x9	4	1	495.9	3x9	5	1	1668.9
ex5_27	3x14	4	3	1.9	3x14	5	3	1.6	3x8	4	1	1275.8	3x8	5	1	1493.5
misex1_02	4x6	5	2	1.1	4x6	5	2	1.0	4x5	5	1	8.1	4x5	6	1	25.2
misex1_06	4x6	4	2	0.8	4x6	5	2	0.8	3x5	4	1	0.7	3x5	4	1	2.0
mp2d_02	3x13	4	3	2.8	3x12	5	3	1.3	3x12	4	1	3340.8	3x12	5	1	3013.7
apex4_015	5x17	8	5	11.7	5x17	9	5	8.8	5x13	8	3	229.6	5x13	9	3	224.1
apex4_017	4x21	8	4	107.3	4x21	8	4	30.2	4x21	8	4	108.9	6x11	9	3	52.0
apex4_018	5x22	8	6	13.1	5x22	9	6	12.2	4x21	8	4	32.0	4x21	9	4	28.2
clip_00	4x27	6	8	21.5	4x27	7	8	7.6	4x21	6	5	27.7	4x18	7	4	41.6
clip_04	4x25	6	8	15.9	4x25	6	8	6.1	4x15	6	3	28.9	4x13	7	3	39.6
sao2_01	7x27	10	8	384.8	7x27	11	8	231.7	7x27	10	8	386.4	7x27	11	8	254.3
sao2_02	3x40	4	8	15.5	4x25	5	8	4.7	3x31	4	4	22.4	4x15	5	2	5964.3
sao2_03	4x34	5	8	23.4	4x32	6	8	10.7	4x26	5	4	334.1	4x23	6	4	1539.7
Avg. (1-8)	39.6	4.4	3.0	2.8	39.3	5.3	3.0	1.5	27.1	4.4	1.3	645.7	26.1	5.3	1.3	780.8
Avg. (9-16)	116.5	6.9	6.9	74.2	113.0	7.6	6.9	39.0	95.4	6.9	4.4	146.3	85.0	7.9	3.9	1018.0
Avg. (1-16)	78.1	5.6	4.9	38.5	76.1	6.4	4.9	20.2	61.3	5.6	2.8	396.0	55.6	6.6	2.6	899.4

TABLE VIII
SUMMARY OF SIMULATION RESULTS OF LATTICE REALIZATIONS.

Instance	JANUS [11]		PHAEDRA $dc = \delta$		PHAEDRA $dc = \delta + 1$	
	delay	power	delay	power	delay	power
5xp1_3	4.16	0.78	1.70	0.81	1.97	0.77
b12_06	2.37	0.69	1.96	0.75	2.14	0.70
dc1_02	1.56	0.64	1.05	0.64	1.24	0.64
ex5_23	1.49	0.74	1.37	0.77	1.38	0.78
ex5_27	1.38	0.76	1.37	0.75	1.37	0.74
misex1_02	1.85	0.68	1.62	0.66	1.57	0.69
misex1_06	2.05	0.65	1.30	0.65	1.27	0.65
mp2d_02	2.37	0.79	1.46	0.84	1.62	0.85
apex4_015	2.59	0.87	2.56	0.98	2.84	0.98
apex4_017	3.38	0.82	2.79	0.96	2.79	0.96
apex4_018	3.49	0.87	2.79	0.96	2.70	0.97
clip_00	2.75	0.85	2.34	0.94	2.39	0.94
clip_04	2.00	0.85	2.10	0.90	2.30	0.88
sao2_01	5.35	0.88	4.10	1.32	4.30	1.37
sao2_02	1.79	0.97	1.70	1.11	1.80	1.03
sao2_03	3.00	1.13	2.10	1.19	2.20	1.11
Avg. (1-8)	2.15	0.72	1.48	0.73	1.57	0.73
Avg. (9-16)	3.04	0.91	2.56	1.05	2.67	1.03
Avg. (1-16)	2.60	0.81	2.02	0.89	2.12	0.88

significantly, increasing the number of switches slightly on small size instances and TROADES can easily handle large size logic functions using less computational effort than PHAEDRA. It was also shown that the realization of a logic function on a lattice with a small number of switches in the critical path can lead to a significant reduction in the delay of the design.

ACKNOWLEDGMENT

This work is supported by the TUBITAK-Career project #113E760 and TUBITAK-2501 project #218E068.

REFERENCES

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[2] Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K.-H. Kim, and C. M. Lieber, "Logic gates and computation from assembled nanowire building blocks," *Science*, vol. 294, no. 5545, pp. 1313–1317, 2001.

[3] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A: Materials Science & Processing*, vol. 80, no. 6, pp. 1165–1172, 2005.

[4] G. Snider, P. Kuekes, T. Hogg, and R. S. Williams, "Nanoelectronic architectures," *Applied Physics A*, vol. 80, no. 6, pp. 1183–1195, 2005.

[5] H. Yan, H. S. Choe, S. Nam, Y. Hu, S. Das, J. F. Klemic, J. C. Ellenbogen, and C. M. Lieber, "Programmable nanowire circuits for nanoprocessors," *Nature*, vol. 470, no. 7333, pp. 240–244, 2011.

[6] G. Gange, H. Søndergaard, and P. J. Stuckey, "Synthesizing optimal switching lattices," *ACM Transactions on Design Automation of Electronic Systems*, vol. 20, no. 1, pp. 6:1–6:14, 2014.

[7] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, and T. Villa, "Logic synthesis for switching lattices by decomposition with p-circuits," in *Euromicro Conference on Digital System Design*, 2016, pp. 423–430.

[8] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Synthesis of switching lattices of dimensional-reducible boolean functions," in *International Conference on Very Large Scale Integration*, 2016, pp. 1–6.

[9] ——, "Composition of switching lattices and autosymmetric boolean function synthesis," in *Euromicro Conference on Digital System Design*, 2017, pp. 137–144.

[10] M. C. Morgul and M. Altun, "Optimal and heuristic algorithms to synthesize lattices of four-terminal switches," *Integration*, vol. 64, pp. 60–70, 2019.

[11] L. Aksoy and M. Altun, "A satisfiability-based approximate algorithm for logic synthesis using switching lattices," in *Design, Automation and Test in Europe Conference*, 2019, pp. 1637–1642.

[12] ——, "Novel methods for efficient realization of logic functions using switching lattices," *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 427–440, 2020.

[13] ——, "A novel method for the realization of complex logic functions using switching lattices," in *International Symposium on Circuits and Systems*, 2020, accepted for publication.

[14] I. Cevik, L. Aksoy, and M. Altun, "Cmos implementation of switching lattices," in *Design, Automation and Test in Europe Conference*, 2020, accepted for publication.

[15] A. S. Cook, "The complexity of theorem-proving procedures," in *ACM Symposium on Theory of Computing*, 1971, pp. 151–158.

[16] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, A. Slisenko, Ed. Nauka, Leningrad. Otdel, 1968, pp. 234–259.

[17] T. Larrabee, "Test pattern generation using boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

[18] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley Publishing Company, 2010.

[19] S. Kang, *Cmos Digital Integrated Circuits: Analysis and Design*, 4th ed. McGraw-Hill Education, 2014.

[20] R. Gupta, B. Tutuianu, and L. T. Pileggi, "The elmore delay as a bound for rc trees with generalized input signals," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 1, pp. 95–104, 1997.

[21] M. Altun and M. Riedel, "Logic synthesis for switching lattices," *IEEE Transactions on Computers*, vol. 61, no. 11, pp. 1588–1600, 2012.
[22] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Springer, 1984.
[23] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solver," in *International Joint Conference on Artifical Intelligence*, 2009, pp. 399–404.
[24] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," MCNC, Tech. Rep., Jan. 1991.

Levent Aksoy received his M.S. and Ph.D. degrees in electronics and communication engineering and electronics engineering from Istanbul Technical University (ITU), Istanbul, Turkey, in 2003 and 2009, respectively. He worked as a post-doctoral researcher in Algorithms for Optimization and Simulation (ALGOS) group of the Instituto de Engenharia de Sistemas e Computadores (INESC-ID), Lisbon. Then, he joined Dialog Semiconductor as a digital design engineer and worked mainly on the design and verification of asynchronous logic circuits. Currently, he is a research fellow in the Emerging Circuits and Computation (ECC) Group in ITU. His research interests include CAD for VLSI circuits with emphasis on solving EDA problems using SAT models and optimization techniques.

Nihat Akkan received his B.Sc. degree in electrical and electronics engineering from Hacettepe University, in 2011 and his M.Sc. degree in electronics engineering from Yıldız Technical University (YTU), in 2015. He is currently pursuing his Ph.D. degree in the Electrical and Electronics Faculty, YTU. He is also a Research Assistant with the Electrical and Electronics Faculty, YTU. His main research areas are compact modeling of organic and inorganic semiconductor devices, solid-state electronics, and circuit and system designs.

Herman Sedef received his B.Sc., M.Sc., and Ph.D. degrees in electronics and communication engineering from Yıldız Technical University (YTU), Istanbul, Turkey, in 1984, 1987, and 1994, respectively. He was a research assistant, from 1986 to 1994. He was an assistant professor and associate professor with the Department of Circuit and Systems, YTU, from 1994 to 2000 and 2000 to 2007, respectively. From 2007, he has been working as a professor in YTU. His research interests are active filter synthesis, filter design, and realization of transfer functions.

Mustafa Altun received his BSc and MSc degrees in electronics engineering at Istanbul Technical University in 2004 and 2007, respectively. He received his PhD degree in electrical engineering with a PhD minor in mathematics at the University of Minnesota in 2012. Since 2013, he has served as an assistant professor at Istanbul Technical University and run the Emerging Circuits and Computation (ECC) Group. Dr. Altun has been served as a principal investigator/researcher of various research projects including EU H2020 RISE, National Science Foundation of USA (NSF) and TUBITAK projects. He is an author of more than 50 peer reviewed papers and a book chapter, and the recipient of the TUBITAK Success, TUBITAK Career, and Werner von Siemens Excellence awards.