# A Satisfiability-Based Approximate Algorithm for Logic Synthesis Using Switching Lattices

Levent Aksoy and Mustafa Altun
*Electronics and Communication Engineering, Istanbul Technical University*
Istanbul, Turkey
{aksoyl, altunmus}@itu.edu.tr

***Abstract*—In recent years the realization of a logic function on two-dimensional arrays of four-terminal switches, called switching lattices, has attracted considerable interest. Exact and approximate methods have been proposed for the problem of synthesizing Boolean functions on switching lattices with minimum size, called lattice synthesis (LS) problem. However, the exact method can only handle relatively small instances and the approximate methods may find solutions that are far from the optimum. This paper introduces an approximate algorithm, called JANUS, that formalizes the problem of realizing a logic function on a given lattice, called lattice mapping (LM) problem, as a satisfiability problem and explores the search space of the LS problem in a dichotomic search manner, solving LM problems for possible lattice candidates. This paper also presents three methods to improve the initial upper bound and an efficient way to realize multiple logic functions on a single lattice. Experimental results show that JANUS can find solutions very close to the minimum in a reasonable time and obtain better results than the existing approximate methods. The solutions of JANUS can also be better than those of the exact method, which cannot be determined to be optimal due to the given time limit, where the maximum gain on the number of switches reaches up to 25%.**

## I. Introduction

Over the years efficient structures, architectures, and techniques have been introduced to implement memory cores, programmable logic arrays and interconnects using nano-technologies [1], [2]. Switching lattices have been proposed as key structures to synthesize logic functions, aiming to achieve significant gains on area, delay, and power consumption with respect to designs using traditional two-terminal switches [3], [4]. From a technological perspective, it is indicated in [3] that switching lattices fit perfectly in emerging technologies, such as nanowire and spin-wave crossbar structures and it is shown in [5] that the conventional CMOS technology can be used to implement switching lattices.

A four-terminal switch is depicted in Fig. 1(a). Its all four terminals are either disconnected (OFF) if its control input $x$ has the value 0, or connected (ON), otherwise. A switching lattice is a network of four-terminal switches, where each switch is connected to its horizontal and vertical neighbors. As an example, Fig. 1(b) depicts the $3\times3$ switching network, where $x_1 \ldots x_9$ denote the control inputs of switches. The lattice function, whose inputs are the control inputs of switches, evaluates to 1 if there is a path between the top and bottom plates of the lattice. Thus, the lattice function is written as the sum of products of control inputs of switches

This work is part of a project that has received funding from the European Union's H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 691178, and supported by the TUBITAK-Career project #113E760.

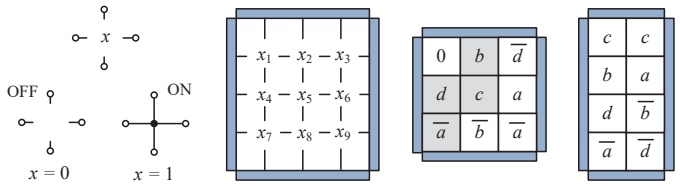


Fig. 1. (a) Four-terminal switch; (b) $3\times3$ four-terminal switching network; realizations of $f=\bar{a}bcd+a\bar{b}c\bar{d}$: (c) using the $3\times3$ switching lattice; (d) using a switching lattice with the minimum size, *i.e.*, $4\times2$.

in each path. The function of the $3\times3$ lattice can be given as $f_{3\times3} = x_1x_4x_7 + x_2x_5x_8 + x_3x_6x_9 + x_1x_4x_5x_8 + x_2x_5x_4x_7 + x_2x_5x_6x_9 + x_3x_6x_5x_8 + x_1x_4x_5x_6x_9 + x_3x_6x_5x_4x_7$. Note that a lattice function is unique and does not include any redundant products, *e.g.*, a possible path $x_1x_2x_3x_6x_9$ in the $3\times3$ switching network is eliminated by the path $x_3x_6x_9$.

A switching lattice can be used to realize a logic function by simply mapping the appropriate literals of this target function and/or constant values (0 and 1) to the control inputs of switches. Thus, the fundamental problem, called lattice mapping (LM), is defined as: given a target function $f$ and an $m\times n$ lattice function $f_{m\times n}$, find the appropriate assignments to the lattice variables such that the target function $f$ can be realized on the $m\times n$ lattice or prove that there exists no such assignment. Note that the LM problem was shown to be *NP-complete* in [6]. As an example, Fig. 1(c) depicts the realization of $f=\bar{a}bcd+a\bar{b}c\bar{d}$ on the $3\times3$ switching lattice.

In the realization of a logic function using a switching lattice, the design complexity is determined as the number of switches, *i.e.*, the lattice size. Thus, the main optimization problem, called the lattice synthesis (LS), is defined as: given the target function $f$, find an $m\times n$ lattice such that there exists an appropriate assignment to the lattice variables, realizing the target function $f$, and $m$ times $n$ is minimum. Returning to our example, Fig. 1(d) presents the realization of $f=\bar{a}bcd+a\bar{b}c\bar{d}$ on a lattice with the minimum size of $4\times2$.

In this paper, we present an approximate method designed for the LS problem, called JANUS, where the LM problem is formulated as a satisfiability (SAT) problem and the initial upper bounds of the search space are improved. We also describe how multiple functions can be realized on a single lattice efficiently. Experimental results show that JANUS can find better solutions than existing methods in less run time.

Rest of the paper is organized as follows. Section II gives the background concepts and related work. Section III presents the proposed algorithm and experimental results are shown in Section IV. Finally, Section V concludes the paper.

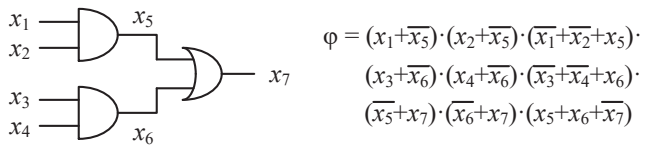


Fig. 2. A combinational network and its POS formula.

## II. Background

### A. Definitions

A Boolean function $f$ in *sum of products* (SOP) form on $r$ variables $x_1,\ldots,x_r$ is a disjunction of $s$ products $p_1,\ldots,p_s$ where a *product* $p_i = l_1 \cdot l_2 \cdot \ldots \cdot l_j$, $i \leq s$ and $j \leq r$, is a conjunction of literals. A *literal* $l_j$, $j \leq r$, is either a variable $x_j$ or its complement $\overline{x_j}$. A product is called *implicant* if and only if it evaluates $f$ to 1 and it is called *prime implicant* if it is implicant and there exist no other implicants 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 $f$ is the maximum number of literals in the products of $f$. Two functions $f$ and $g$ are said to be dual functions if $f(x_1,\ldots,x_r) = \overline{g}(\overline{x_1},\ldots,\overline{x_r})$.

A Boolean function $\varphi$ in *product of sums* (POS) form on $r$ variables is a conjunction of $t$ clauses $c_1,\ldots,c_t$ where a *clause* $c_i = l_1 + l_2 + \ldots + l_j$, $i \leq t$ and $j \leq r$, is a disjunction of literals. Note that 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 $\varphi$ in POS form that makes $\varphi$ to be equal to 1 or to prove that $\varphi$ is equal to 0.

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 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 [7]. Fig. 2 shows a combinational circuit and its formula in the POS form.

In a switching lattice, a *four-connected path* is a sequence of switches connected by taking horizontal and vertical moves and an *eight-connected path* is generated by also taking diagonal moves. Recall that a lattice function includes all the irredundant four-connected paths between the top and bottom plates. It is shown in [3] that the dual of a lattice function consists of all the irredundant eight-connected paths between the left and right plates. Returning to our example in Fig. 1, the dual of $f_{3\times3}$ consists of 17 products[1]. Thus, finding a realization of a target function on an $m \times n$ switching lattice considering the four-connected paths between the top and bottom plates can also be done by finding a realization of the dual of the target function on the $m \times n$ lattice considering the eight-connected paths between the left and right plates. Hence, we take into account these two possible realizations of the target function. Thus, algorithms, that can find a lattice function and its dual in ISOP form, are also developed.

[1]They are $x_1x_2x_3$, $x_1x_2x_6$, $x_1x_5x_3$, $x_1x_5x_6$, $x_1x_5x_9$, $x_4x_2x_3$, $x_4x_2x_6$, $x_4x_5x_3$, $x_4x_5x_6$, $x_4x_5x_9$, $x_4x_8x_6$, $x_4x_8x_9$, $x_7x_5x_3$, $x_7x_5x_6$, $x_7x_5x_9$, $x_7x_8x_6$, and $x_7x_8x_9$.

TABLE I
Number of products in the $m \times n$ lattice function and its dual.

| m/n | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 2 | 2<br>4 | 3<br>8 | 4<br>16 | 5<br>32 | 6<br>64 | 7<br>128 | 8<br>256 |
| 3 | 4<br>7 | 9<br>17 | 16<br>41 | 25<br>99 | 36<br>239 | 49<br>577 | 64<br>1393 |
| 4 | 6<br>10 | 17<br>28 | 36<br>78 | 67<br>216 | 118<br>600 | 203<br>1666 | 344<br>4626 |
| 5 | 10<br>13 | 37<br>41 | 94<br>139 | 205<br>453 | 436<br>1497 | 957<br>4981 | 2146<br>16539 |
| 6 | 16<br>16 | 77<br>56 | 236<br>250 | 621<br>1018 | 1668<br>4286 | 4883<br>18730 | 14880<br>81192 |
| 7 | 26<br>19 | 163<br>73 | 602<br>461 | 1905<br>2439 | 6562<br>13833 | 26317<br>86963 | 110838<br>539537 |
| 8 | 42<br>22 | 343<br>92 | 1528<br>872 | 5835<br>6004 | 25686<br>45788 | 139231<br>421182 | 797048<br>3779226 |

Table I presents the number of products in the $m \times n$ lattice function and its dual at the top and bottom of each entry, respectively, where $2 \leq m,n \leq 8$. Observe that as the number of rows and columns increases, the number of products in the lattice functions increases dramatically, pointing out the lattices that can be used to realize a rich variety of logic functions. Note that a lattice function may have fewer or more products than its dual, *e.g.*, $2 \times 4$ and $8 \times 4$ lattices. For the lattices with sizes very close to each other, there exists a wide range of functions with different number of products. For example, while $f_{6\times6}$ contains 1668 products, $f_{7\times5}$ has 1905 products. This is also true for the lattices with the same size. As an example, while $f_{3\times8}$ includes 64 products, $f_{6\times4}$ has 236 products. It should be indicated that not only the number of products, but also the number of literals in each product is important while realizing a function on a switching lattice.

### B. Related Work

In [3], an upper bound on the lattice size is obtained by finding the common literals in the products of the target function and its dual, and the lower bound on the lattice size is computed based on the minimum degrees of the target function and its dual. The exact algorithm of [6] explores the search space using a binary search technique in between the lower and upper bounds computed in [3]. During this search, LM problems are solved for the given target function and possible lattices. The LM problem is encoded as a quantified Boolean formula (QBF) problem, the QBF constraints are converted to SAT clauses, and a solution is found using a SAT solver. The approximate method of [6] restricts this exact QBF formulation by allowing the paths to include only the literals in the given products, reducing the size of SAT problems. However, since the approximate method may yield a non-optimal solution, it may solve more LM problems than the exact method. The method of [8] synthesizes the D-reducible form of a target function, which is composed of two small sub-functions, on a switching lattice. In [8], these sub-functions are synthesized using the exact method of [6] and then, their solutions are merged into a single lattice. Note that not every logic function can be represented in the D-reducible form. Similarly, the methods of [9], [10] exploit the p-circuits and autosymmetric form of a target function, respectively and use the algorithms of [3], [6] to find a solution on the decomposed

smaller functions. In [10], the target function is synthesized with multiple lattices sharing the common ones, but adding extra logic gates which may not be desirable due to the wires between these gates and lattice control inputs. The method of [11] determines a number of promising lattice candidates and uses a method of [6] to find if one of these lattices leads to a solution.

## III. The Proposed Aprroximate Algorithm

JANUS takes the target function $f$ as an input and finds its implementation on a switching lattice as described below:

1) Determine the *lower bound* (*lb*) and *upper bound* (*ub*) of the LS problem in terms of the number of switches.
2) If $lb = ub$, return the solution found in computing *ub*.
3) Determine the *middle point* as $mp = \lfloor (lb+ub)/2 \rfloor$ and generate a set of lattice candidates as done in [6].
4) For each lattice candidate, generate the related LM problem and check if $f$ can be realized using the lattice candidate. If there exists a solution to the LM problem, set *ub* to *mp* and go to Step 6.
5) If there are no solutions for all lattice candidates, set *lb* to $mp+1$.
6) If $lb < ub$, go to Step 3. Otherwise, return the solution.

In this section, we first present the SAT encoding of the LM problem (Step 4), and then, introduce the methods, that improve the initial upper bounds of the LS problem, where JANUS is also used (Step 1). Finally, we describe how JANUS can be applied to multiple functions efficiently.

### A. Finding a Solution to the LM Problem

Given the target function with a minimum number of products obtained using a logic minimization tool and the lattice function, all in ISOP form, initially, the structural check is performed. In this case, we check for each product of the target function if there exists a product in the lattice function with a number of literals greater than or equal to that of the product in the target function. For our example in Fig. 1, neither $f_{8\times1} = x_1x_2x_3x_4x_5x_6x_7x_8$ nor $f_{2\times4} = x_1x_5 + x_2x_6 + x_3x_7 + x_4x_8$ can be used to realize $f = \overline{a}bcd + a\overline{b}c\overline{d}$. Because while $f_{8\times1}$ and $f$ have 1 and 2 products, respectively, $f_{2\times4}$ and $f$ have products with 2 and 4 literals, respectively. The same check is also carried out for the duals of the target and lattice functions.

If the structural check is passed, then we check if there exists an assignment to the lattice function variables included in the lattice variable set $LV$ from the target literal set $TL$ consisting of the target function literals and constants 0 and 1 such that every entry in the truth table of the target function is satisfied. The LM problem is encoded as a SAT problem in three steps as follows.

First, we generate the sets $LV$ and $TL$ and the mapping variables $lv_i\_tl_j$, where $lv_i \in LV$ with $1 \le i \le |LV|$ and $tl_j \in TL$ with $1 \le j \le |TL|$, and $|A|$ denotes the cardinality of set $A$. 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 in Fig. 1(c), $LV = \{x_1, x_2, \ldots, x_9\}$, $TL = \{a, \overline{a}, b, \overline{b}, c, d, \overline{d}, 0, 1\}$, and as an example, the mapping variable $x_1\_\overline{a}$ indicates that the lattice variable $x_1$ is assigned to $\overline{a}$, if $x_1\_\overline{a}$ is set to high. Moreover, we generate clauses, which confirm that each lattice variable is assigned to only one element in the $TL$ set, 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 $\sum$ are AND and OR operators, respectively. While the former clauses confirm that for each lattice variable, at least one of the mapping variables should be set to high, the latter ones ensure that for each lattice variable, when one mapping variable is set to high, the other ones should be set to low.

Second, to satisfy the target function, for each truth table entry, we generate the combinational circuit corresponding to the lattice function and assign the target function value at this entry to the circuit output. The circuit inputs, *i.e.*, the lattice function variables, are associated with the truth table entry and denoted as $lv_{i\_tte}$, where $1 \le i \le |LV|$ and *tte* is the truth table entry. We obtain the POS formula of the circuit as shown in Fig. 2 and simplify it based on the logic value at the circuit output. For our example in Fig. 1(c), Fig. 3 presents the circuits generated for the $abcd = 0000$ and $abcd = 0111$ points, where the target function is 0 and 1, respectively. For the sake of clarity, only three products of $f_{3\times3}$ are shown in this figure.

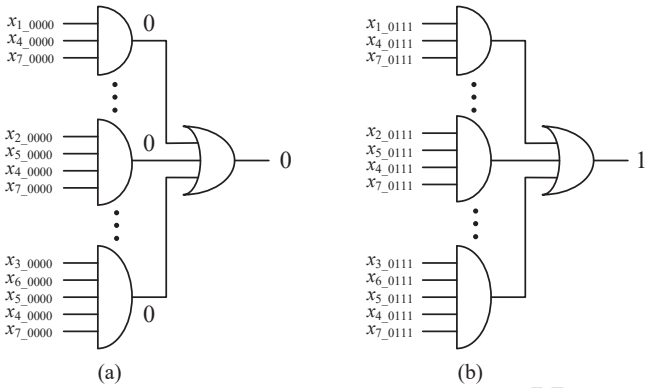


Fig. 3. The combinational circuits of $f_{3\times3}$ for $f = \overline{a}bcd + a\overline{b}c\overline{d}$: (a) when $abcd = 0000$ and $f$ is low; (b) when $abcd = 0111$ and $f$ is high.

Observe from Fig. 3(a) that when the target function is low for a truth table entry, the logic 0 at the OR gate output can be propagated to the outputs of AND gates and thus, the POS formula of the circuit can be reduced to the only ones that indicate the possible ways of setting each AND gate output to 0. For our example in Fig. 3(a), the clause generated for the AND gate at the top is $\overline{x_{1\_0000}} + \overline{x_{4\_0000}} + \overline{x_{7\_0000}}$.

Observe from Fig. 3(b) that when the target function is high for a truth table entry, the clauses, which ensure that if an input of the OR gate is high, the OR gate output should be high, can be removed from the POS formula. Moreover, for this case, there should be at least one four-connected path in between the top and bottom plates where the control inputs of associated switches are set to high. This path is shown on the lattice of Fig. 1(c) with the shaded blocks when $abcd = 0111$. There exist two facts related to this case described as follows: i) in each row of the switching lattice, there should be at least one

switch whose control input has a high value; ii) in each two consecutive rows, there should be at least one situation that the control inputs of switches on the same column have a high value. We generate clauses for these facts to hold.

In order to link the mapping variables to the circuit inputs, for each mapping variable, we generate clauses which ensure that when a mapping variable is set to high, the associated circuit input, *i.e.*, a lattice variable, is set to a value determined by the truth table entry. For our example, when $abcd = 0000$, the constraints, such as, $x_1\_a \Rightarrow \overline{x_{1\_0000}}$ and $x_3\_\overline{b} \Rightarrow x_{3\_0000}$, ensure that the circuit input has the corresponding value when a lattice variable is assigned to a target literal. Note that $\Rightarrow$ is the imply operator and $a \Rightarrow b$ is equal to $\overline{a}+b$. When a lattice variable is assigned to a constant value 0 or 1, the related circuit input is set to that value.

Third, if the degree of the target function, denoted as $\delta$, is equal to that of the lattice function, for each product with $\delta$ literals in the target function, we generate clauses indicating that at least one of the products with $\delta$ variables in the lattice function should be used to realize this product. Consider the realization of $f = \overline{b}\overline{c}\overline{d} + abcde$ on the $3\times3$ lattice, where $\delta$ is 5. It is obvious that the product $x_1x_4x_5x_6x_9$ or $x_3x_6x_5x_4x_7$ of $f_{3\times3}$ should realize $abcde$. Among many possibilities, this can be achieved by setting the mapping variables $x_1\_a$, $x_4\_b$, $x_5\_c$, $x_6\_d$, and $x_9\_e$ to high. Moreover, it was noticed that realizing products with a large number of literals in the lattice is a hard task. Hence, if a product of a target function has more than 5 (determined empirically) literals, we generate clauses which ensure that this product is realized by at least one product with more than 5 variables in the lattice function.

Thus, a SAT problem, that formalizes the LM problem, is generated based on the target and lattice functions. We also consider the realization of the dual of the target function using the dual of the given lattice function and generate another SAT problem using a formulation similar to the one given above. This is due to the fact that the dual of the lattice function may have less number of products than that of the lattice function as shown in Table I and the dual of the target function may have less number of truth table entries where the target function is high, yielding a SAT problem with a small number of variables and clauses. After generating the alternative SAT problem, we apply a SAT solver to the one with the least complexity computed as the number of variables times the number of clauses. Since it is easier for the SAT solver to find a solution if it exists than to prove that there is no solution, 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.

Although there are benchmarks that JANUS can handle, there are still instances that it may find them hard to solve. However, such instances can be solved using a divide and conquer approach where JANUS is applied to sub-functions including a small number of products. Such a method is used to find an upper bound on the LS problem as described next.

### *B. Computing the Initial Lower and Upper Bounds*

We find the initial lower bound of the LS problem by taking into account the products of the lattice and target functions. Starting from the lattice size $s$ equal to 1, we obtain all possible lattice candidates for the lattice size $s$ and for each lattice candidate, we check if all products of the target function are covered by the products of this lattice function as done in the structural check described in Section III-A. If it is so, the structural check is performed on the duals of the target and lattice functions. If the structural check is also passed in this case, the lower bound is determined by the lattice size. Otherwise, another lattice candidate is tried. If there exist no lattices with the lattice size $s$ that pass the structural check, then $s$ is increased by 1 and this procedure is repeated until the lower bound is found. It was observed that this computation may yield better values than the general technique given in [3].

There exist three efficient methods used to find an initial upper bound. The dual production (DP) [3] method realizes a target function using an $m\times n$ lattice, where $n$ and $m$ are the number of products in the target function and its dual, respectively. In the product separation (PS) method [6], the $n$ products of a target function are placed on the columns of a lattice each separated by a column full of zeros, filling the unspecified entries of the lattice with constant 1. Thus, a solution with a $\delta\times(2n-1)$ lattice is found, where $\delta$ is the degree of the target function. In the dual product separation (DPS) method [11], the $m$ products of the dual of the target function are placed on the rows of a lattice separated by a row full of ones, filling the unspecified entries of the lattice with constant 0. Thus, a solution with a $(2m-1)\times\gamma$ lattice is found, where $\gamma$ is the degree of the dual of the target function.

However, the improved version of the PS method, called IPS, can be obtained by reducing the number of isolation columns between the products as follows: i) the products with a single literal can be used as isolation columns between the products, if this literal is placed on every row of that column; ii) the products with two literals do not need isolation columns, if one literal is placed on the $\delta^{th}$ row and the other is placed on the other rows of that column; iii) for each product with more than 2 literals, we check if it can be realized with another product on a $\delta\times2$ lattice without using an isolation column. This check is passed if the number of products in the dual of a function, including only these two products, is less than or equal to $\delta$. Similarly, the improved version of the DPS method, called IDPS, can be obtained.

Furthermore, we developed the divide and synthesize (DS) method which includes three steps described as follows: 1) partition the products in the target function $f$ into two sub-functions $g$ and $h$ such that $f = g+h$, where $g$ and $h$ have a number of products close to each other and a minimum number of literals; 2) apply JANUS to these sub-functions and add its solutions into a lattice using a single isolation column. As an example, suppose that the sub-functions are realized using $5\times3$ and $2\times4$ lattices. Thus, a $5\times8$ lattice is required to realize $f$; 3) for each solution on the sub-functions, explore

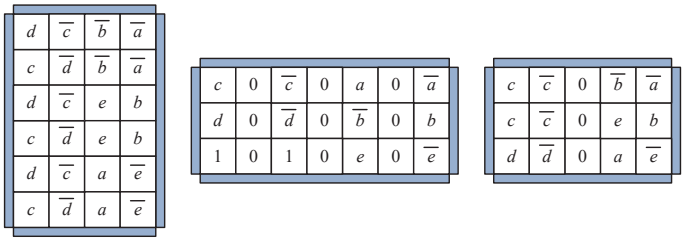


Fig. 4. Upper bounds of $f = cd + \bar{c}\bar{d} + a\bar{b}e + \bar{a}b\bar{e}$: (a) DP; (b) PS; (c) IPS.

alternative realizations with a smaller number of rows and columns. This is based on the fact that any logic function can be realized using a lattice with a number of rows greater than 2. Initially, on the lattice obtained at the second step, we determine its size and number of rows, denoted as the best cost *bc* and best row *br*, respectively. Then, if $br > 2$, we check if the final lattice can have a size less than *bc* as described in the following procedure: i) for a solution of a sub-function with a $br \times n$ lattice, where $br > 3$, check if a $(br-1) \times k$ lattice, where $k > n$, can be used to synthesize this sub-function. Note that $k$ initially set to $n$ is incremented by 1 till the *bc* value is exceeded or a solution is found; ii) for a solution of a sub-function with an $m \times n$ lattice, where $1 < m < br-1$, check if this sub-function can be realized using an $(br-1) \times k$ lattice, where $k < n$. Note that $k$ initially set to $n$ is decremented by 1 till there exists no solution. At the end of this procedure, if a solution with a size less than *bc* is found, the lattice is updated and this procedure is iterated till *br* is 3.

Thus, in JANUS, the initial upper bound is computed as the minimum of solutions of all these methods.

As an example, consider $f = cd + \bar{c}\bar{d} + a\bar{b}e + \bar{a}b\bar{e}$. The DP (Fig. 4(a)), PS (Fig. 4(b)), and DPS methods find a solution with a $6 \times 4$, $3 \times 7$, and $11 \times 4$ lattice on this target function, respectively. Besides, the solutions of the proposed IPS (Fig. 4(c)), IDPS, and DS methods are $3 \times 5$, $8 \times 4$, and $3 \times 5$ lattices, respectively. Thus, the initial upper bound is computed as 15. Note that the initial lower bound is 12 and the minimum solution is obtained using a $3 \times 4$ lattice.

### C. Realizing Multiple Functions on a Single Lattice

To realize multiple logic functions on a single lattice, we developed a method, called JANUS-MF, which is similar to the DS algorithm described in Section III-B. It is composed of two parts. In the first part, as done in the second step of the DS method, we find the realization of each logic function using JANUS and add this solution into a single lattice, separating it from another with a column full of zeros, filling the unspecified entries with constant 1. In the second part, we check if each logic function can be synthesized using a lattice with a smaller number of rows and columns as done in the third step of the DS method.

## IV. Experimental Results

This section presents the results of JANUS, JANUS-MF, and the methods of [6], [9], [11]. Note that JANUS, developed in Perl, uses *espresso* [12] to find the ISOP forms of target functions and their duals and *glucose4.1* [13] to solve a SAT problem. The proposed algorithms can be found at http://www.ecc.itu.edu.tr/images/d/d4/JANUS.zip. 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 [11]. The results of the method [9] were taken from [11]. In methods of [9], [11], the exact algorithm of [6] was used since its solutions yield better results than its approximate version.

Table II presents the results of algorithms on 48 instances, where *#in*, *#pi*, and $\delta$ denote the number of inputs, prime implicants, and degree of the target functions in ISOP form, respectively. In this table, *lb* stands for the lower bound found as described in Section III-B, *oub* is the old upper bound computed based on the DP, PS, and DPS methods [11], and *nub* is the new upper bound found considering also the solutions of the IPS, IDPS, and DS methods. Finally, *sol* and *CPU* denote the solution and run time of algorithms in seconds, respectively. Note that the algorithms were run on an Intel Xeon CPU at 2.40GHz with 28 cores and 128GB RAM with the CPU time limit of 6 hours.

Observe from Table II that the use of new methods introduced for finding an upper bound improves the existing upper bound of [11] by 42.8% on average, reducing the search space of the LS problem significantly using a little computational effort. Note that while the DP, PS, and DPS methods find a smaller upper bound on only one instance than other methods, the proposed IPS, IDPS, and DS methods lead to better upper bounds on 39 instances than other methods. Observe also that the new upper bound can be better than the solutions of existing methods proposed for the LS problem, *e.g.*, 5xp1_3.

Observe from Table II that JANUS can find better solutions in terms of lattice size than the exact algorithm of [6], *e.g.*, ex5_15, ex5_17, and ex5_24, since it explores a smaller search space due to the improved upper bounds and it encodes the LM problem as a SAT problem efficiently. Moreover, the solutions of JANUS on all instances are smaller than or equal to those found by the existing algorithms, having the smallest lattice size on average. Furthermore, JANUS finds a solution using the least computational effort on average. On the other hand, the strict rules on the realization of a product in the approximate method of [6] yield the worst solutions on instances ex5_15, ex5_17, and ex5_23. The solutions of the heuristic method of [11] may be far away from the optimal, *e.g.*, 5xp1_3, ex5_24, and mp2d_01, since it does not consider all possible lattice candidates. The method of [9] leads to solutions which are worse than those of other algorithms on average.

Table III presents the results of JANUS-MF on three LGS91 benchmarks [14], where *#out* denotes the number of outputs of given instances, *size* stands for the number of switches in the lattice, and the *straight-forward method* is actually the first part of JANUS-MF where the solution of JANUS on each logic function is merged into a single lattice as described in Section III-C. Observe that JANUS-MF outperforms the straight-forward method by searching for realizations with a small number of rows and columns, where the maximum gain is achieved as 32% on the bw instance.

TABLE II

SUMMARY OF INITIAL LOWER AND UPPER BOUNDS AND RESULTS OF ALGORITHMS ON SINGLE FUNCTIONS.

| Instance | Function Details | | | Initial Lower and Upper Bounds | | | | [9] | [11] | | Approximate [6] | | Exact [6] | | JANUS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #in | #pi | $\delta$ | lb | oub | nub | CPU | sol | sol | CPU | sol | CPU | sol | CPU | sol | CPU |
| 5xp1_1 | 7 | 11 | 5 | 16 | 105 | 32 | 4.1 | 5x10 | 5x5 | 501.2 | 6x5 | 21600.0 | 5x5 | 21600.0 | 4x6 | 2023.2 |
| 5xp1_3 | 6 | 14 | 5 | 15 | 135 | 40 | 57.3 | 4x11 | 5x27 | 21600.0 | 11x4 | 21600.0 | 11x4 | 21600.0 | 4x9 | 19745.8 |
| b12_00 | 6 | 4 | 4 | 9 | 24 | 20 | 0.2 | 4x3 | 4x3 | 0.3 | 4x3 | 0.6 | 4x3 | 2.1 | 4x3 | 0.3 |
| b12_01 | 7 | 7 | 4 | 12 | 35 | 20 | 0.2 | 4x4 | 4x4 | 1.1 | 4x4 | 1.6 | 5x3 | 8.5 | 5x3 | 1.1 |
| b12_02 | 8 | 7 | 5 | 12 | 42 | 24 | 0.8 | 5x8 | 4x4 | 5.7 | 5x4 | 3.7 | 4x4 | 35.4 | 4x4 | 4.1 |
| b12_03 | 4 | 4 | 2 | 6 | 6 | 6 | 0.1 | 2x5 | 3x2 | 0.1 | 3x2 | 0.2 | 3x2 | 0.1 | 3x2 | 0.1 |
| b12_06 | 9 | 9 | 6 | 15 | 44 | 24 | 4.3 | 5x4 | 5x4 | 23.8 | 5x4 | 4.6 | 5x4 | 139.3 | 5x4 | 23.8 |
| b12_07 | 7 | 6 | 4 | 16 | 24 | 24 | 0.3 | 6x8 | 3x6 | 1.1 | 5x4 | 2.5 | 3x6 | 5.4 | 3x6 | 1.5 |
| c17_01 | 4 | 4 | 2 | 6 | 6 | 6 | 0.1 | 3x2 | 3x2 | 0.1 | 3x2 | 0.2 | 3x2 | 0.1 | 3x2 | 0.1 |
| clpl_00 | 7 | 4 | 4 | 12 | 16 | 15 | 0.2 | 4x5 | 3x4 | 0.4 | 3x4 | 0.3 | 3x4 | 1.3 | 3x4 | 0.3 |
| clpl_03 | 11 | 6 | 6 | 16 | 36 | 24 | 0.6 | 6x9 | 3x6 | 19.6 | 3x6 | 2.3 | 3x6 | 200.0 | 3x6 | 84.9 |
| clpl_04 | 9 | 5 | 5 | 15 | 25 | 18 | 0.3 | 5x8 | 3x5 | 5.0 | 3x5 | 1.3 | 3x5 | 25.3 | 3x5 | 1.3 |
| dc1_00 | 4 | 4 | 3 | 9 | 16 | 15 | 0.2 | 4x4 | 3x3 | 0.1 | 3x3 | 0.4 | 3x3 | 0.4 | 3x3 | 0.2 |
| dc1_02 | 4 | 4 | 3 | 12 | 16 | 15 | 0.2 | 3x5 | 3x4 | 0.1 | 3x4 | 0.3 | 4x3 | 0.2 | 4x3 | 0.3 |
| dc1_03 | 4 | 4 | 4 | 9 | 20 | 18 | 0.2 | 4x5 | 4x3 | 0.2 | 4x3 | 0.4 | 4x3 | 0.5 | 4x3 | 0.3 |
| ex5_06 | 7 | 8 | 3 | 16 | 32 | 24 | 0.3 | 3x10 | 3x6 | 1.2 | 3x7 | 12.0 | 3x6 | 7.2 | 3x6 | 2.1 |
| ex5_07 | 8 | 10 | 4 | 24 | 40 | 27 | 0.7 | 3x13 | 4x6 | 19.7 | 3x9 | 332.2 | 4x6 | 473.2 | 3x8 | 2.5 |
| ex5_08 | 8 | 7 | 3 | 20 | 21 | 21 | 0.2 | 3x9 | 3x7 | 0.0 | 3x7 | 9.3 | 3x7 | 51.2 | 3x7 | 7.2 |
| ex5_09 | 8 | 10 | 4 | 24 | 40 | 30 | 12.3 | 3x11 | 4x6 | 5.7 | 3x8 | 108.2 | 4x6 | 454.6 | 3x8 | 17.6 |
| ex5_10 | 6 | 7 | 3 | 16 | 21 | 21 | 0.2 | 3x9 | 3x6 | 0.7 | 3x6 | 1.4 | 3x6 | 3.8 | 3x6 | 0.5 |
| ex5_12 | 8 | 9 | 3 | 15 | 25 | 20 | 0.2 | 5x9 | 3x5 | 1.8 | 3x5 | 1.7 | 3x5 | 13.7 | 3x5 | 12.6 |
| ex5_13 | 8 | 9 | 3 | 24 | 36 | 27 | 0.9 | 3x13 | 3x8 | 10.0 | 4x6 | 57.6 | 4x6 | 190.2 | 3x8 | 2.8 |
| ex5_14 | 8 | 8 | 2 | 16 | 16 | 16 | 0.2 | 3x11 | 2x8 | 0.9 | 2x8 | 1.2 | 2x8 | 6.7 | 2x8 | 0.2 |
| ex5_15 | 8 | 12 | 4 | 20 | 72 | 33 | 3.1 | 4x13 | 4x7 | 48.5 | 6x12 | 21600.0 | 6x5 | 21600.0 | 3x8 | 2562.4 |
| ex5_17 | 8 | 14 | 4 | 20 | 105 | 42 | 23.2 | 4x10 | 4x7 | 1425.6 | 10x6 | 21600.0 | 6x6 | 21600.0 | 3x9 | 4377.6 |
| ex5_19 | 8 | 6 | 3 | 16 | 18 | 18 | 0.1 | 5x7 | 3x6 | 1.4 | 3x6 | 1.1 | 3x6 | 6.9 | 3x6 | 0.4 |
| ex5_21 | 8 | 10 | 3 | 20 | 57 | 30 | 0.5 | 4x9 | 3x7 | 8.2 | 4x7 | 1364.6 | 3x7 | 280.9 | 3x7 | 790.8 |
| ex5_22 | 7 | 6 | 3 | 16 | 33 | 21 | 0.2 | 3x8 | 3x6 | 1.3 | 3x6 | 2.0 | 3x6 | 8.4 | 3x6 | 1.2 |
| ex5_23 | 8 | 12 | 4 | 24 | 92 | 36 | 39.0 | 4x11 | 4x8 | 2465.0 | 11x5 | 21600.0 | 3x9 | 15418.6 | 3x9 | 3726.4 |
| ex5_24 | 8 | 14 | 5 | 20 | 105 | 33 | 7.0 | 5x14 | 15x7 | 21600.0 | 3x11 | 21600.0 | 4x7 | 21600.0 | 3x8 | 1638.8 |
| ex5_25 | 8 | 8 | 3 | 20 | 40 | 27 | 0.3 | 3x8 | 3x7 | 16.4 | 3x7 | 6.4 | 3x7 | 79.4 | 3x7 | 152.7 |
| ex5_26 | 8 | 10 | 3 | 20 | 57 | 30 | 0.7 | 4x11 | 3x7 | 12.9 | 3x9 | 384.5 | 3x7 | 238.5 | 3x7 | 36.3 |
| ex5_27 | 8 | 11 | 4 | 20 | 77 | 27 | 1.3 | 4x10 | 4x6 | 58.1 | 3x8 | 1049.5 | 4x6 | 1561.3 | 3x8 | 1229.3 |
| ex5_28 | 8 | 9 | 3 | 24 | 27 | 27 | 0.2 | 3x13 | 3x8 | 5.3 | 3x8 | 180.2 | 6x4 | 51.5 | 3x8 | 1.6 |
| misex1_00 | 4 | 2 | 4 | 6 | 8 | 8 | 0.1 | 4x3 | 4x2 | 0.1 | 4x2 | 0.2 | 4x2 | 0.2 | 4x2 | 0.1 |
| misex1_01 | 6 | 5 | 4 | 12 | 35 | 18 | 0.2 | 5x5 | 3x5 | 1.9 | 4x4 | 1.7 | 3x5 | 7.4 | 3x5 | 1.1 |
| misex1_02 | 7 | 5 | 5 | 12 | 40 | 25 | 0.4 | 5x5 | 5x4 | 24.0 | 5x4 | 4.6 | 5x4 | 50.9 | 5x4 | 19.7 |
| misex1_03 | 7 | 4 | 5 | 9 | 28 | 20 | 0.3 | 4x6 | 4x3 | 0.9 | 5x3 | 1.2 | 4x3 | 3.9 | 4x3 | 0.5 |
| misex1_04 | 4 | 5 | 4 | 12 | 25 | 18 | 0.2 | 4x7 | 3x4 | 0.2 | 5x3 | 1.0 | 3x4 | 0.7 | 3x4 | 0.4 |
| misex1_05 | 6 | 6 | 4 | 12 | 42 | 21 | 0.3 | 4x6 | 4x4 | 4.6 | 5x4 | 4.9 | 4x4 | 13.4 | 4x4 | 2.1 |
| misex1_06 | 6 | 5 | 4 | 12 | 35 | 18 | 0.2 | 4x7 | 5x3 | 1.3 | 5x3 | 1.6 | 5x3 | 4.7 | 5x3 | 1.3 |
| misex1_07 | 6 | 4 | 4 | 9 | 20 | 18 | 0.3 | 5x5 | 4x3 | 0.7 | 5x3 | 1.0 | 4x3 | 1.6 | 4x3 | 0.5 |
| mp2d_01 | 10 | 8 | 5 | 24 | 48 | 30 | 4.3 | 4x11 | 5x7 | 28.7 | 4x7 | 291.3 | 3x9 | 6478.3 | 3x9 | 3257.3 |
| mp2d_02 | 11 | 10 | 4 | 28 | 50 | 33 | 0.9 | 4x13 | 4x9 | 33.9 | 4x7 | 730.7 | 4x7 | 4580.7 | 4x7 | 948.9 |
| mp2d_03 | 10 | 5 | 8 | 15 | 72 | 32 | 4.5 | 7x6 | 5x5 | 42.3 | 4x6 | 188.2 | 6x4 | 1322.7 | 4x6 | 271.2 |
| mp2d_04 | 10 | 6 | 9 | 15 | 57 | 36 | 5.5 | 7x3 | 7x3 | 18.9 | 7x3 | 58.8 | 7x3 | 3043.1 | 7x3 | 286.8 |
| mp2d_06 | 5 | 3 | 5 | 8 | 18 | 16 | 0.3 | 5x4 | 6x2 | 0.3 | 7x2 | 1.2 | 4x3 | 1.1 | 6x2 | 0.4 |
| newtag_00 | 8 | 8 | 3 | 16 | 32 | 24 | 0.2 | 3x8 | 3x6 | 2.7 | 3x6 | 2.1 | 3x6 | 19.0 | 3x6 | 2.2 |
| Average | 7.2 | 7.3 | 4.0 | 15.5 | 41.1 | 23.5 | 3.7 | 32.1 | 22.7 | 1000.0 | 22.0 | 2800.4 | 18.9 | 2974.8 | 18.3 | 859.2 |

TABLE III

SUMMARY OF RESULTS OF ALGORITHMS ON MULTIPLE FUNCTIONS.

| Instance | #out | straight-forward method | | | JANUS-MF | | |
|---|---|---|---|---|---|---|---|
| | | sol | size | CPU | sol | size | CPU |
| bw | 28 | 5x119 | 595 | 12.7 | 3x135 | 405 | 14.1 |
| misex1 | 7 | 5x31 | 155 | 25.3 | 3x42 | 126 | 30.4 |
| squar5 | 8 | 5x31 | 155 | 31.7 | 3x36 | 108 | 59.7 |

## V. CONCLUSIONS

This paper introduced the approximate method JANUS for the synthesis of logic functions using lattices with a small number of switches. It presented an efficient encoding of the LM problem as a SAT problem, introduced methods that improve the existing initial upper bounds of the search space, and described how JANUS can be used to realize multiple functions on a single lattice. It was shown that JANUS can find promising solutions with respect to the existing methods.

## REFERENCES


[1] A. Dehon, "Nanowire-based programmable architectures," *ACM JECT*, vol. 1, pp. 109–162, 2005.

[2] M. Dong and L. Zhong, "Nanowire crossbar logic and standard cell-based integration," *IEEE TVLSI*, vol. 17, no. 8, pp. 997–1007, 2009.

[3] M. Altun and M. Riedel, "Logic synthesis for switching lattices," *IEEE Transactions on Computers*, vol. 61, pp. 1588–1600, 2012.

[4] D. Alexandrescu, M. Altun, L. Anghel, A. Bernasconi, V. Ciriani, L. Frontini, and M. Tahoori, "Logic synthesis and testing techniques for switching nano-crossbar arrays," *MICPRO*, vol. 54, pp. 14–25, 2017.

[5] S. Safaltin, O. Gencer, M. C. Morgul, L. Aksoy, S. Gurmen, C. A. Moritz, and M. Altun, "Realization of four-terminal switching lattices: Technology development and circuit modeling," in *DATE*, 2019.

[6] G. Gange, H. Søndergaard, and P. J. Stuckey, "Synthesizing optimal switching lattices," *ACM TODAES*, vol. 20, pp. 6:1–6:14, 2014.

[7] T. Larrabee, "Test pattern generation using boolean satisfiability," *IEEE TCAD*, vol. 11, no. 1, pp. 4–15, 1992.

[8] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Synthesis of switching lattices of dimensional-reducible boolean functions," in *VLSI-SoC*, 2016, pp. 1–6.

[9] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, and T. Villa, "Logic synthesis for switching lattices by decomposition with p-circuits," in *DSD*, 2016, pp. 423–430.

[10] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Composition of switching lattices for regular and for decomposed functions," *MICPRO*, vol. 60, pp. 207–218, 2018.

[11] M. Morgul and M. Altun, "Optimal and heuristic algorithms to synthesize lattices of four-terminal switches," *Integration*, in press.

[12] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Springer, 1984.

[13] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solver," in *IJCAI*, 2009, pp. 399–404.

[14] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," MCNC, Tech. Rep., Jan. 1991.