A Novel Method for the Realization of Complex Logic Functions Using Switching Lattices

Levent Aksoy and Mustafa Altun
Department of Electronics and Communication Engineering, Istanbul Technical University
34469, Maslak, Istanbul, Turkey
Email: {aksoyl, altunmus}@itu.edu.tr

***Abstract*—Over the years, efficient algorithms have been proposed to realize logic functions on two-dimensional arrays of four-terminal switches, called switching lattices, using the fewest number of switches. Although existing algorithms can easily find a solution on logic functions with a small number of inputs and products, they can hardly handle large size instances. In order to cope with such logic functions, in this paper, we introduce SISYPHUS that exploits Boolean decomposition techniques and incorporates a state-of-art algorithm designed for the realization of logic functions using switching lattices. Experimental results indicate that SISYPHUS can find competitive solutions on logic functions with a small number of inputs and products when compared to those of previously proposed algorithms. Moreover, its solutions on large size functions are obtained using a little computational effort and are significantly better than the best solutions found so far.**

I. INTRODUCTION

In recent years, the realization of logic functions using switching lattices has attracted a significant amount of interest since lattices offer reconfigurability, a rich variety of possible implementations, and an alternative way to realize and synthesize logic functions [1], [2]. A switching lattice is a two-dimensional network of four-terminal switches where each switch is connected to its horizontal and vertical neighbors. A four-terminal switch has a control input x and four terminals. If the control input has a value 0, all terminals are disconnected. Otherwise, they are connected. Fig. 1(a) shows the behavior of the four-terminal switch and Fig. 1(b) depicts the 3×3 switching network where $x_1, \ldots, x_9$ denote the control inputs of four-terminal switches. In a lattice with four-terminal switches, a path is defined as a sequence of switches connected by taking horizontal and vertical moves. The function of an $m \times n$ lattice $f_{m \times n}$, whose inputs are the control inputs of switches, evaluates to 1 if there is a path between the top and bottom plates and can be written as the sum of products of control inputs of switches in each path. Fig. 1(c) presents the lattice function $f_{3 \times 3}$. A lattice function is unique and does not include any redundant products. As an example, a possible path $x_1x_2x_5x_8$ in the 3×3 lattice is eliminated by $x_2x_5x_8$.

A logic function can be realized using a switching lattice by finding appropriate assignments to the control inputs of switches from the literals of the logic function and/or constant 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 the appropriate assignments to the control inputs of switches such that f can be realized on the $m \times n$ lattice or prove that there exists no such assignment. The LM problem is an NP-complete problem [3]. As an example, consider the realization of $f(y_1, y_2, y_3, y_4) = \sum(2, 5, 7, 10, 12, 13, 14)$, which can be written as $f = \overline{y_1}y_2y_4 + y_1y_2\overline{y_3} + y_1y_2\overline{y_4} + \overline{y_2}y_3\overline{y_4}$, using the 3×4 lattice. Fig. 2(a) shows the realization of f on the given lattice[1]. However, the design complexity in the realization of a logic function using a lattice is defined as the number of four-terminal switches, *i.e.*, lattice size. Thus, the main optimization problem, called *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 f, and m times n is minimum. Fig. 2(b) presents the realization of the logic function f using a lattice with a minimum size, *i.e.*, 3×3 lattice[2].

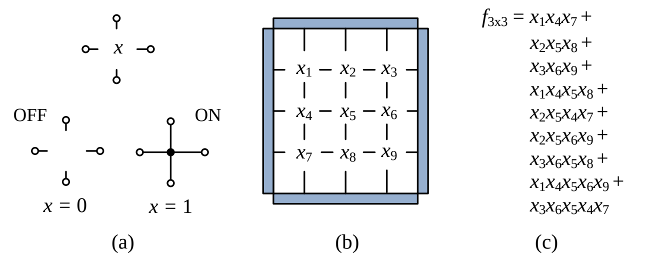


Fig. 1. (a) Four-terminal switch; (b) the 3×3 four-terminal switching network; (c) the 3×3 switching lattice function.

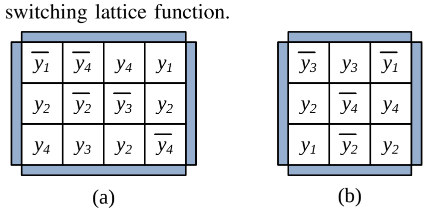


Fig. 2. Realizations of $f = \overline{y_1}y_2y_4 + y_1y_2\overline{y_3} + y_1y_2\overline{y_4} + \overline{y_2}y_3\overline{y_4}$ using switching lattices: (a) 3×4; (b) 3×3.

Exact and efficient approximate algorithms [3]–[5] were introduced to realize logic functions on switching lattices using the fewest number of switches. However, they cannot handle large size logic functions which include a high

[1]Considering all the paths between the top and bottom plates and applying the $y \cdot y = y$ and $y \cdot \overline{y} = 0$ laws, the logic function realized by the lattice can be written as $g = \overline{y_1}y_2y_4 + \overline{y_2}y_3\overline{y_4} + y_2\overline{y_3}y_4 + y_1y_2\overline{y_4} + y_1y_2\overline{y_3}$. Also, $y_2\overline{y_3}y_4$ can be eliminated since it is covered by $y_1y_2\overline{y_3}$ and $\overline{y_1}y_2y_4$.

[2]Considering all the paths between the top and bottom plates and applying the $y \cdot y = y$ and $y \cdot \overline{y} = 0$ laws, the logic function realized by the lattice can be written as $h = y_1y_2\overline{y_3} + \overline{y_2}y_3\overline{y_4} + \overline{y_1}y_2y_4 + y_1y_2y_3\overline{y_4}$. Also, $y_1y_2y_3\overline{y_4}$ can be reduced to $y_1y_2\overline{y_4}$ due to the $y_1y_2\overline{y_3}$ product.

number of inputs and products since the problem complexity increases dramatically as the number of inputs and products in the target and lattice function increases. Also, alternative approaches [6]–[8] were proposed, where a logic function is decomposed into smaller sub-functions, their realizations on a lattice are found using the previously proposed algorithms [1], [3], and these realizations are merged into a single lattice. However, they also cannot cope with such large size instances and obtain poor results since they decompose a logic function only once, use only a single decomposition method, and do not explore alternative realizations of these sub-functions. Moreover, the divide and conquer method of [9] iteratively decomposes a logic function into sub-functions until they can be handled by the algorithm of [5] easily, but it uses a single decomposition method. Hence, in this paper, we introduce SISYPHUS that can find a solution to a large size logic function using three decomposition techniques in order to determine the best sub-functions that may lead to a design with a small number of four-terminal switches. It also decomposes logic functions into smaller sub-functions until their realizations on a switching lattice can be found easily by the state-of-art algorithm of [5]. After the sub-functions are determined and their realizations are found, it considers alternative realizations of these sub-functions that may reduce the number of switches in the final design. Experimental results show that SISYPHUS can find significantly better solutions on large size logic functions using less computational effort than existing algorithms.

The rest of this paper is organized as follows: Section II gives the background concepts and related work. Section III introduces SISYPHUS and Section IV presents the experimental results. Finally, Section V concludes the paper.

II. BACKGROUND

A. Preliminaries

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* $p_i = l_1 \cdot l_2 \cdot \ldots \cdot l_j$, $i \leq s$ and $j \leq r$, is a conjunction of literals. A *literal* l_j, $j \leq r$, is either a variable 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 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.

B. Related Work

The exact method of [3] explores the search space of the LS problem in a dichotomic search manner in between the lower and upper bounds computed in [1]. For each possible lattice, an LM problem is encoded as a quantified Boolean formula (QBF) problem, the QBF constraints are converted to satisfiability (SAT) clauses, and a solution is found using a SAT solver. The algorithm of [5] 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 an efficient SAT encoding for the LM problem. The method of [4] determines a number of promising lattice candidates and uses an algorithm of [3] to find if one of these lattices leads to a solution. The methods of [6], [7], and [8] 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. Similarly, the divide and conquer method of [9] decomposes a target function into sub-functions iteratively.

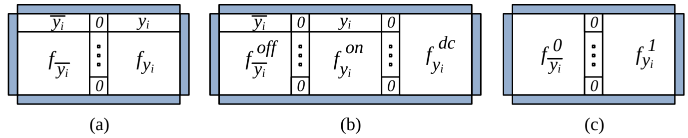


Fig. 3. Realizations of logic function decompositions on a single lattice: (a) $f = \overline{y_i} f_{\overline{y_i}} + y_i f_{y_i}$ (b) $f = \overline{y_i} f_{\overline{y_i}}^{off} + y_i f_{y_i}^{on} + f_{y_i}^{dc}$ (c) $f = f_{\overline{y_i}}^{0} + f_{y_i}^{1}$.

III. THE PROPOSED ALGORITHM

SISYPHUS takes the target function as an input and returns its realization on a switching lattice as an output. Its main steps are given as follows:

1) Compute the initial lower bound as described in [5] and the upper bound using the techniques proposed in [5], except the DS method.
2) If the difference between the computed upper and lower bound, *dulb*, is less than or equal to 31, find the realization of the target function using the method of [5].
3) Otherwise, decompose it iteratively into sub-functions.
4) Find the realizations of these sub-functions using the algorithm of [5], merge them into a single lattice, and compute the lattice size.
5) Explore alternative realizations of sub-functions which may reduce the final lattice size.

Based on our experience with the algorithm of [5], it can handle logic functions with a *dulb* value less than or equal to 31 easily. Hence, we determine the *dulb* value as 31 empirically. In the following subsections, the last three steps of SISYPHUS are described in detail.

A. Decompositions of a Logic Function

SISYPHUS exploits three decomposition techniques. The first one is the Shannon expansion. For a variable y_i, $i \leq r$, a logic function is written as $f = \overline{y_i} f_{\overline{y_i}} + y_i f_{y_i}$, where $f_{\overline{y_i}}$ and f_{y_i} are the negative and positive co-factors obtained when y_i is set to logic 0 and 1 in f, respectively. For our example in Fig. 2, the logic function f decomposed over y_1 is written as $f = \overline{y_1}(y_2 y_4 + \overline{y_2} y_3 \overline{y_4}) + y_1(y_2 \overline{y_3} + y_3 \overline{y_4})$.

Fig. 3(a) presents the realization of the first decomposition using a single lattice. Assume that the sub-functions $f_{\overline{y_i}}$ and f_{y_i} are realized using an $m_{f_{\overline{y_i}}} \times n_{f_{\overline{y_i}}}$ and $m_{f_{y_i}} \times n_{f_{y_i}}$ lattice, respectively. Thus, the first decomposition requires a lattice of $max(1 + m_{f_{\overline{y_i}}}, 1 + m_{f_{y_i}}) \times (1 + n_{f_{\overline{y_i}}} + n_{f_{y_i}})$.

In the second decomposition, the ISOP form of the logic function f is found. For a variable y_i, $i \leq r$, the logic function is written as $f = \overline{y_i} f_{\overline{y_i}}^{off} + y_i f_{y_i}^{on} + f_{y_i}^{dc}$, where $f_{\overline{y_i}}^{off}$ and $f_{y_i}^{on}$ consist of the products in the ISOP form of f including the $\overline{y_i}$ and y_i literals which are set to logic 0 and 1, respectively, and $f_{y_i}^{dc}$ consists of the products in the ISOP form of f that do not include any literal of y_i. For our example in Fig. 2, $f = \overline{y_1}y_2y_4 + y_1y_2\overline{y_3} + y_1y_2\overline{y_4} + \overline{y_2}y_3\overline{y_4}$ decomposed over y_1 is written as $f = \overline{y_1}(y_2y_4) + y_1(y_2\overline{y_3} + y_3\overline{y_4}) + (\overline{y_2}y_3\overline{y_4})$.

Fig. 3(b) presents the realization of the second decomposition using a single lattice. Assuming that the sub-functions $f_{\overline{y_i}}^{off}$, $f_{y_i}^{on}$, and $f_{y_i}^{dc}$ are realized using an $m_{f_{\overline{y_i}}^{off}} \times n_{f_{\overline{y_i}}^{off}}$, $m_{f_{y_i}^{on}} \times n_{f_{y_i}^{on}}$, and $m_{f_{y_i}^{dc}} \times n_{f_{y_i}^{dc}}$ lattice, respectively, the second decomposition needs a $max(1 + m_{f_{\overline{y_i}}^{off}}, 1 + m_{f_{y_i}^{on}}, m_{f_{y_i}^{dc}}) \times (2 + n_{f_{\overline{y_i}}^{off}} + n_{f_{y_i}^{on}} + n_{f_{y_i}^{dc}})$ lattice.

In the third decomposition, the ISOP form of the logic function f is also used. For a variable y_i, $i \leq r$, the logic function is written as $f = f_{\overline{y_i}}^0 + f_{y_i}^1$, where $f_{\overline{y_i}}^0$ and $f_{y_i}^1$ initially consist of all the products of the ISOP form including the $\overline{y_i}$ and y_i literals, respectively. Then, the products of the ISOP form of f, that do not include any literal of y_i, are one by one added into either $f_{\overline{y_i}}^0$ or $f_{y_i}^1$, favoring the one that has the smallest number of products. When these functions have the same number of products, the product is added into the one which has the maximum number of common literals according to literals of the product. For our example in Fig. 2, $f = \overline{y_1}y_2y_4 + y_1y_2\overline{y_3} + y_1y_2\overline{y_4} + \overline{y_2}y_3\overline{y_4}$ decomposed over y_1 is written as $f = (\overline{y_1}y_2y_4 + \overline{y_2}y_3\overline{y_4}) + (y_1y_2\overline{y_3} + y_1y_3\overline{y_4})$.

Fig. 3(c) presents the realization of the third decomposition using a single lattice. Assuming that the sub-functions $f_{\overline{y_i}}^0$ and $f_{y_i}^1$ are respectively realized using an $m_{f_{\overline{y_i}}^0} \times n_{f_{\overline{y_i}}^0}$ and $m_{f_{y_i}^1} \times n_{f_{y_i}^1}$ lattice, the third decomposition requires a $max(m_{f_{\overline{y_i}}^0}, m_{f_{y_i}^1}) \times (1 + n_{f_{\overline{y_i}}^0} + n_{f_{y_i}^1})$ lattice.

Observe from Fig. 3 that the first and second decompositions generate at most two and three sub-functions to be realized, respectively, requiring one and two isolation columns full of logic 0. Note that both first and second decompositions have a single literal y_i on the first row of the lattice and the generated sub-functions do not include any literal of y_i. The third decomposition generates at most two sub-functions which include a literal y_i. Note that the third decomposition generates only one sub-function if the logic function f is always true when y_i is equal to logic 0 or 1. In this case, the generated sub-function is actually the logic function f itself and hence, this decomposition is not taken into consideration. In the same case, the first and second decompositions generate only one sub-function as well. But, they are regarded as valid, since the sub-function does not include any literal of y_i and hence, it is different from the logic function itself. Although there exist cases, where these decompositions lead to the same lattice realizations, they generally generate different sub-functions that yield realizations with different complexities.

The procedure for the decomposition of a logic function f can be given as follows: For each decomposition technique and each variable in f, y_i, $i \leq r$, i) find the decomposition of f over y_i; ii) extract the sub-functions in the decomposition; iii) estimate the lattice size of sub-functions with the IPS method of [5] used to find an improved upper bound on the lattice size; iv) find the size of the lattice realizing the decomposition as described above and keep the decomposition whose realization has the smallest lattice size and store its sub-functions. This procedure is repeated until each sub-function has a *dulb* value less than or equal to 31.

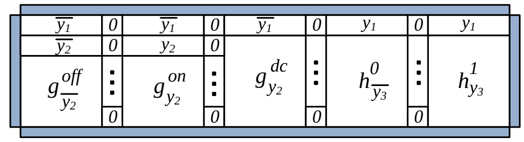


Fig. 4. Realization of a decomposed target function $f = \overline{y_1}\,\overline{y_2}g_{\overline{y_2}}^{off} + \overline{y_1}y_2g_{y_2}^{on} + \overline{y_1}g_{y_2}^{dc} + y_1h_{\overline{y_3}}^0 + y_1h_{y_3}^1$ on a single lattice.

B. Finding the Realizations of Decomposed Functions

After the logic functions are decomposed into sub-functions, the target function is written in the SOP form where each product is a conjunction of sub-functions and/or literals. As an example, assume that the target function is decomposed using the first technique and the variable y_1 as $f = \overline{y_1}f_{\overline{y_1}} + y_1f_{y_1}$. Assume also that the sub-functions $f_{\overline{y_1}}$, denoted as g, and f_{y_1}, denoted as h, are decomposed using the second and third decompositions and the variables y_2 and y_3, respectively. Thus, the target function is given as $f = \overline{y_1}(\overline{y_2}g_{\overline{y_2}}^{off} + y_2g_{y_2}^{on} + g_{y_2}^{dc}) + y_1(h_{\overline{y_3}}^0 + h_{y_3}^1) = \overline{y_1}\,\overline{y_2}g_{\overline{y_2}}^{off} + \overline{y_1}y_2g_{y_2}^{on} + \overline{y_1}g_{y_2}^{dc} + y_1h_{\overline{y_3}}^0 + y_1h_{y_3}^1$. The realizations of sub-functions are found using the algorithm of [5], they are merged into a single lattice, and the design complexity of this lattice is computed. Fig. 4 presents the realization of our decomposed function on a single lattice.

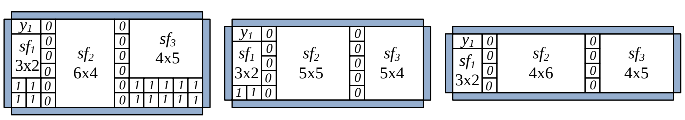
Assume that the target function has k products, and consequently, k sub-functions, denoted as sf_i, $i \leq k$, and the number of literals in each product is denoted as lc_i. Also, suppose that each sub-function is realized using an $m_{sf_i} \times n_{sf_i}$ lattice. Thus, the design complexity of the lattice realizing the target function, denoted as dc, is computed as $max_i(lc_i + m_{sf_i})$ times $(k - 1 + \sum_i n_{sf_i})$.

C. Exploring Alternative Realizations of Sub-Functions

The lattice size can be further reduced exploring alternative realizations of sub-functions. Considering the target function expressed in the SOP form with k products at the previous step, the row of the associated lattice, *i.e.*, $max_i(lc_i + m_{sf_i})$, where $i \leq k$, is denoted as the maximum row mr. If $mr > 2$, alternative realizations of sub-functions are checked as follows: i) for each product including a sub-function sf_i and lc_i literals, where $lc_i + m_{sf_i} = mr$, $i \leq k$, if $m_{sf_i} > 3$, check if an $(m_{sf_i} - 1) \times c$ lattice, where $c > n_{sf_i}$, can be used to synthesize sf_i. Note that c initially set to n_{sf_i} is incremented by 1 till the dc value is exceeded or a solution is found; ii) for each product including a sub-function sf_i and lc_i literals, where $lc_i + m_{sf_i} < mr$, $i \leq k$, check if sf_i can be realized using an $(mr - lc_i - 1) \times c$ lattice, where $c < n_{sf_i}$. Note that c initially set to n_{sf_i} is decremented by 1 till there exists no solution. At the end of this procedure, the lattice design complexity is computed as described in Section III-B. If it is smaller than dc, the final lattice and the realizations of sub-functions are updated. If mr is equal to $max_i(lc_i) + 3$, where $i \leq k$, this procedure is terminated. Otherwise, mr is decremented by 1 and this procedure is repeated. These alternative realizations are checked by the algorithm of [5].

TABLE I
SUMMARY OF RESULTS OF ALGORITHMS ON SMALL AND LARGE SIZE LOGIC FUNCTIONS.

Instance	Function Details		[6]		[4]		Exact [3]		JANUS [5]		MEDEA [9]		SISYPHUS	
	in	pi	sol	CPU	sol	CPU	sol	CPU	sol	CPU	sol	CPU	sol	CPU
5xp1_1	7	11	5x10	4.2	5x5	501.2	5x5	21600.0	4x6	2023.2	4x8	2.2	4x8	5.5
5xp1_3	6	14	4x11	11.1	5x27	21600.0	11x4	21600.0	4x9	19745.8	5x8	55.8	4x10	106.7
apex4_16	9	11	5x11	2742.5	8x21	21600.0	8x21	21600.0	4x12	21600.0	5x12	14.2	6x11	14.6
apex4_17	9	12	4x22	7419.7	8x23	21600.0	8x23	21600.0	7x7	21600.0	5x15	8.6	7x11	71.5
apex4_18	9	14	22x14	21600.0	43x5	21600.0	8x27	21600.0	7x8	21600.0	6x13	859.4	7x10	28.7
ex5_15	8	12	4x13	2.2	4x7	48.5	6x5	21600.0	3x8	2562.4	3x11	5.4	3x12	19.2
ex5_17	8	14	4x13	21.6	4x7	1425.6	6x6	21600.0	3x9	4377.6	4x10	29.1	4x9	13.9
ex5_23	8	12	4x11	13.2	4x8	2465.0	3x9	15418.6	3x9	3726.4	3x12	29.9	3x12	24.2
ex5_27	8	11	4x11	7.8	4x6	58.1	4x6	1561.3	3x8	1229.3	3x9	1.7	3x10	5.7
inc_03	7	11	6x8	37.4	5x21	21600.0	19x3	21600.0	4x9	15023.7	4x11	2.6	5x8	61.0
mp2d_03	10	5	7x6	19.8	5x5	42.3	6x4	1322.7	4x6	271.2	4x8	5.5	5x7	9.4
rd53_01	5	16	4x11	16.5	5x31	21600.0	9x5	21600.0	4x11	21600.0	4x11	52.5	4x11	47.5
sao2_02	10	22	5x15	266.8	23x7	21600.0	4x43	21600.0	4x13	21600.0	4x18	11.3	4x19	16.9
alu4_02	14	50	25x54	21600.0	79x7	21600.0	6x99	21600.0	59x7	21600.0	5x36	164.2	5x37	179.3
alu4_03	14	72	56x73	21600.0	143x7	21600.0	7x143	21600.0	107x7	21600.0	5x64	850.1	8x32	1214.3
alu4_05	14	90	90x93	21600.0	179x9	21600.0	9x179	21600.0	136x9	21600.0	5x121	2736.0	7x71	3946.6
alu4_06	14	36	36x36	21600.0	71x7	21600.0	7x71	21600.0	55x7	21600.0	5x35	1285.0	6x25	509.5
apex4_01	9	33	28x35	21600.0	73x6	21600.0	9x65	21600.0	55x6	21600.0	6x36	214.5	7x33	901.4
apex4_03	9	69	47x69	21600.0	137x8	21600.0	9x137	21600.0	103x8	21600.0	6x78	1094.0	7x70	344.9
apex4_04	9	76	45x81	21600.0	147x8	21600.0	9x151	21600.0	111x8	21600.0	6x96	1583.4	7x81	193.2
apex4_06	9	76	46x85	21600.0	8x151	21600.0	8x151	21600.0	8x114	21600.0	5x112	1862.0	7x73	254.2
apex4_07	9	75	46x79	21600.0	143x8	21600.0	9x149	21600.0	108x8	21600.0	6x89	975.2	7x74	279.4
apex4_08	9	76	46x84	21600.0	149x8	21600.0	8x151	21600.0	113x8	21600.0	6x87	2758.3	7x75	816.9
apex4_09	9	72	45x75	21600.0	8x143	21600.0	8x143	21600.0	8x108	21600.0	5x104	275.4	7x73	309.2
apex4_10	9	74	41x79	21600.0	131x8	21600.0	9x147	21600.0	100x8	21600.0	5x98	761.6	7x69	631.7
Z9sym	9	84	34x112	21600.0	143x7	21600.0	6x167	21600.0	107x7	21600.0	5x112	1938.7	6x79	2177.3
Avg. (1-13)	8.0	12.7	72.8	2474.1	98.8	11980.1	80.9	18023.3	36.2	12073.8	47.2	82.9	47.5	32.7
Avg. (14-26)	10.5	67.9	3446.3	21600.0	1008.4	21600.0	1085.1	21600.0	762.0	21600.0	440.5	1269.1	415.2	904.5
Avg. (1-26)	9.3	40.3	1759.5	12037.0	553.6	16790.0	583.0	19811.6	399.1	16836.9	243.8	676.0	231.4	468.6



(a) (b) (c)

Fig. 5. Realizations of sub-functions on alternative lattices leading to designs with different complexities: (a) 6×13; (c) 5×13; (a) 4×15.

As a simple example, assume that a target function is decomposed into sub-functions, their realizations are found using the algorithm of [5], and are merged into a single lattice as shown in Fig. 5(a). The design complexity of this lattice is computed as 6×13, *i.e.*, 78. However, assume that the sub-function sf_2 and sf_3 can be realized using a 5×5 and 5×4 lattice, respectively, as shown in Fig. 5(b). Thus, the design complexity reduces to 65. Moreover, assume that the sub-function sf_2 can be realized using a 4×6 lattice as shown in Fig. 5(c). Thus, the design complexity reduces to 60.

IV. EXPERIMENTAL RESULTS

In this section, we present the results of SISYPHUS and the methods of [3]–[6], [9]. Note that SISYPHUS, developed in Perl, uses *espresso* [10] to find the ISOP forms of logic functions. We used 26 instances which were taken from [11] and categorized into two classes. The first (second) class, *i.e.*, the first (last) 13 instances, consists of small (large) size logic functions, where the number of inputs times the number of products is less than or equal to 220 (greater than or equal to 297). Table I presents the function details and the results of methods, where *in* and *pi* denote the number of inputs and prime implicants of the target functions in ISOP form, respectively. Also, *sol* and *CPU* stand for the solution of methods and their run-time in seconds, respectively. All methods were run on an Intel Xeon processor at 2.40GHz with 28 cores and 128GB RAM under a time limit of 6 hours.

Observe from Table I that the solutions of SISYPHUS on small size instances are close to those found by the algorithms of [5] and [9] and are better than those of algorithms [3], [4], [6] on average. Also, SISYPHUS finds the solutions of small size instances using less computational effort than the previously proposed algorithms on average. Existing methods, except the algorithm of [9], cannot cope with large size instances and their solutions are obtained with their methods used to find an upper bound. On the other hand, the solutions of SISYPHUS on these instances are obtained using the least computational effort and are significantly better than those of the existing algorithms on average. This experiment also shows that the use of three different decomposition techniques generally leads to better solutions in terms of the number of switches and run-time when compared to algorithms including only one decomposition technique [6], [9].

V. CONCLUSIONS

This paper introduced an efficient method for the realization of complex logic functions on a switching lattice that existing algorithms find them hard to handle. Experimental results clearly indicated that its solutions on small size instances are very competitive to those found using the previously proposed algorithms and its solutions on large size instances are significantly better than the best solutions found so far.

ACKNOWLEDGMENT

This work is supported by the European Union's H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 691178 and the TUBITAK-2501 project #218E068.

References

[1] M. Altun and M. Riedel, "Logic synthesis for switching lattices," *IEEE Transactions on Computers*, vol. 61, no. 11, pp. 1588–1600, 2012.

[2] 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," *Microprocessors and Microsystems*, vol. 54, pp. 14–25, 2017.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] ——, "Composition of switching lattices for regular and for decomposed functions," *Microprocessors and Microsystems*, vol. 60, pp. 207–218, 2018.

[9] L. Aksoy and M. Altun, "Novel methods for efficient realization of logic functions using switching lattices," *IEEE Transactions on Computers*, 2019, accepted for publication.

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

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