CMOS Implementation of Switching Lattices

Ismail Cevik, Levent Aksoy, and Mustafa Altun

***Abstract*—Switching lattices consisting of four-terminal switches are introduced as area-efficient structures to realize logic functions. Many optimization algorithms have been proposed, including exact ones, realizing logic functions on lattices with the fewest number of four-terminal switches, as well as heuristic ones. Hence, the computing potential of switching lattices has been justified adequately in the literature. However, the same thing cannot be said for their physical implementation. There have been conceptual ideas for the technology development of switching lattices, but no concrete and directly applicable technology has been proposed yet. In this study, we show that switching lattices can be directly and efficiently implemented using a standard CMOS process. To realize a given logic function on a switching lattice, we propose static and dynamic logic solutions. The proposed circuits as well as the compared conventional ones are designed and simulated in the Cadence environment using TSMC 65nm CMOS process. Experimental post layout results on logic functions show that switching lattices occupy much smaller area than those of the conventional CMOS implementations, while they have competitive delay and power consumption values.**

***Index Terms*—switching lattice, four-terminal switch, psuedo NMOS logic, dynamic logic, 65nm CMOS technology.**

I. Introduction

A switching lattice, formed as a two dimensional network of four-terminal switches, is introduced as a crossbar based, regular, dense, and area-efficient structure for logic computing [1]. A four-terminal switch, corresponding to a crossbar cross-point, has one control input x and four terminals. As shown in Fig. 1(a), all of its terminals are either disconnected (OFF), if its control input has the value 0 or connected (ON), otherwise. A 3×3 switching lattice is shown in Fig. 1(b), where $x_1 \ldots x_9$ denote the control inputs of switches. The logic function for the lattice evaluates to 1 if there is a path between the top and bottom plates of the lattice, so it can be found by taking the sum of the products (SOP) of the control inputs along each path. Fig. 1(c) shows the logic function $f_{3\times3}$ for the lattice given in Fig. 1(b).

In order to implement a target logic function, literals of the function as well as constant logic values (0 and 1) are mapped to the control inputs of switches such that the lattice and target functions are equal to each other. Here, the main goal is finding the minimum lattice size, i.e., the minimum number of four-terminal switches. To achieve this goal, many different algorithms have been introduced [1]–[9].

To compare the realization of a logic function in terms of the number of two-terminal and four-terminal switches, consider $f(a,b,c,d) = abc + \overline{a}\overline{b}c + a\overline{c}\overline{d} + \overline{a}c\overline{d}$ in SOP

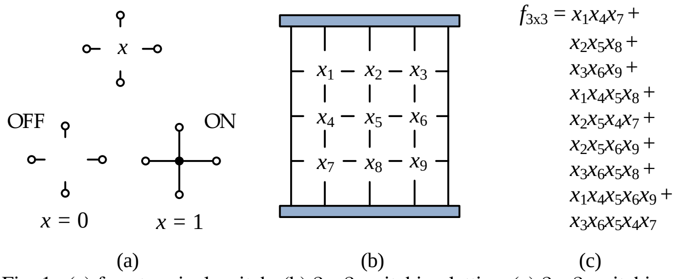
x
OFF
ON
x = 0
x = 1
x1 x2 x3
x4 x5 x6
x7 x8 x9
f3x3 = x1x4x7 + x2x5x8 + x3x6x9 + x1x4x5x8 + x2x5x4x7 + x2x5x6x9 + x3x6x5x8 + x1x4x5x6x9 + x3x6x5x4x7
(a)
(b)
(c)

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

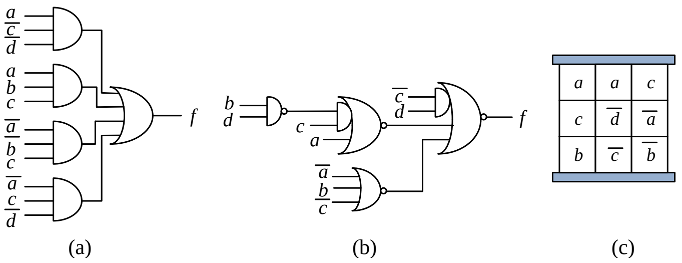

Fig. 2. Different realizations for $f = abc + \overline{a}\overline{b}c + a\overline{c}\overline{d} + \overline{a}c\overline{d}$: (a) two-level synthesis using two-terminal switches (MOS transistors); (b) multi-level synthesis aiming to minimize the number of two-terminal switches; (c) JANUS [8] aiming to minimize the number of four-terminal switches.

form as an example. This function can be realized using AND and OR gates as shown in Fig. 2(a). Note that this straight-forward implementation requires 42 MOS transistors, *i.e.*, two-terminal switches, without counting the ones for the inverters of primary inputs. However, the number of MOS transistors can be reduced further by applying a state-of-art logic synthesis tool to the logic function with a number of synthesis scripts. Fig. 2(b) presents an optimized solution with 24 MOS transistors, not counting the ones for the inverters of primary inputs. On the other hand, as shown in Fig. 2(c), our logic function can be realized using a 3×3 lattice, which requires 9 four-terminal switches, found using the publicly available algorithm of [8], called JANUS in the paper.

This example, supported by many other inspiring examples and results in the literature [1]–[9], clearly shows the computing potential of switching lattices as well as their area efficiency. However, no concrete and directly applicable technology has been yet proposed for the implementation of switching lattices. In [1], physical formations of nanowire and magnetic four-terminal switches are conceptually given, but they lack details on simulation and fabrication. By using three dimensional (3D) technology computer-aided design (TCAD) simulations, it is shown in [10] that a four-terminal switch can be implemented with CMOS technology. However, the four-terminal switch device structures of [10] have a lower capability of conducting current while having nearly the same area when compared to our proposed implementation. Moreover, the devices of [10] cannot be simulated with transistor models provided by the semiconductor foundries.

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-2501 project #218E068.

The authors are with the Department of Electronics and Communication Engineering, Istanbul Technical University, Maslak, 34469, Istanbul, Turkey (e-mail: ceviki, aksoyl, altunmus@itu.edu.tr).

In this study, we introduce the implementation of a four-terminal switch and a switching lattice in a standard CMOS process. We also describe the realization of logic functions on switching lattices using static pseudo NMOS logic and dynamic logic. We present the results of designs realizing logic functions using conventional two-terminal switches and the proposed four-terminal switches under TSMC 65nm CMOS process. It is observed that the implementations with switching lattices offer around at least 2X smaller area than those of the conventional implementations with comparable delay and power consumption values. This performance improvement is based on two factors: 1) the number of four-terminal switches needed to synthesize a logic function is smaller than those of two-terminal switches; and 2) switching lattices have dense, regular, and metal-connection-free layouts.

This paper is organized as follows. Section II describes the implementation of a four-terminal switch and a switching lattice in a standard CMOS process. Section III describes the realization of logic functions on switching lattices using static pseudo NMOS logic and dynamic logic. Section IV presents the experimental results and Section V concludes the paper with discussions and future directions.

II. CMOS Implementation of Four-Terminal Switch and Switching Lattice

Previously, two CMOS-compatible four-terminal switch devices, namely the square-shaped and the cross-shaped, are introduced in [10]. Since the square-shaped one has higher current conducting capability, smaller area, and it is easier to manufacture compared to the square-shaped one, we decide to further investigate it in this study. Fig. 3(a) and (b) present 3D view of its implementation and its layout in TSMC 65nm process as an n-type switch, respectively. In Fig. 3(a), gate insulator may be silicon oxide or an other insulator depending on the technology node used, and STI stands for the shallow trench isolation. In Fig. 3(b), the implemented switch obeys all the design rules and fits in a compact area; its pitch is only 50% larger than the minimum size NMOS transistor in 65nm process. Also, a switching lattice can be easily built with these switches by connecting them with shared diffusion regions without a need of metal contacts. The only metal connection in the switch structure is the gate contact shown in the upper right corner of the square gate shown in Fig. 3(b).

However, this design has two main drawbacks. First, the gate extension mandated by the design rules results in very long channels. The minimum channel length of the four-terminal switch is approximately equal to three times the gate width of a minimum size NMOS transistor. The minimum size four-terminal switch in 65nm process ends up having a width to length ratio of 120nm/400nm. The resulting conductivity of the equivalent switch is less than one sixth of that in a minimum size NMOS transistor having a width to length ratio of 120nm/60nm. The second drawback is that this square-shaped device as well as the cross-shaped one cannot be simulated with CMOS transistor models.

To overcome these drawbacks, we propose a four-terminal switch device. Fig. 3(c) and (d) show its 3D view and layout

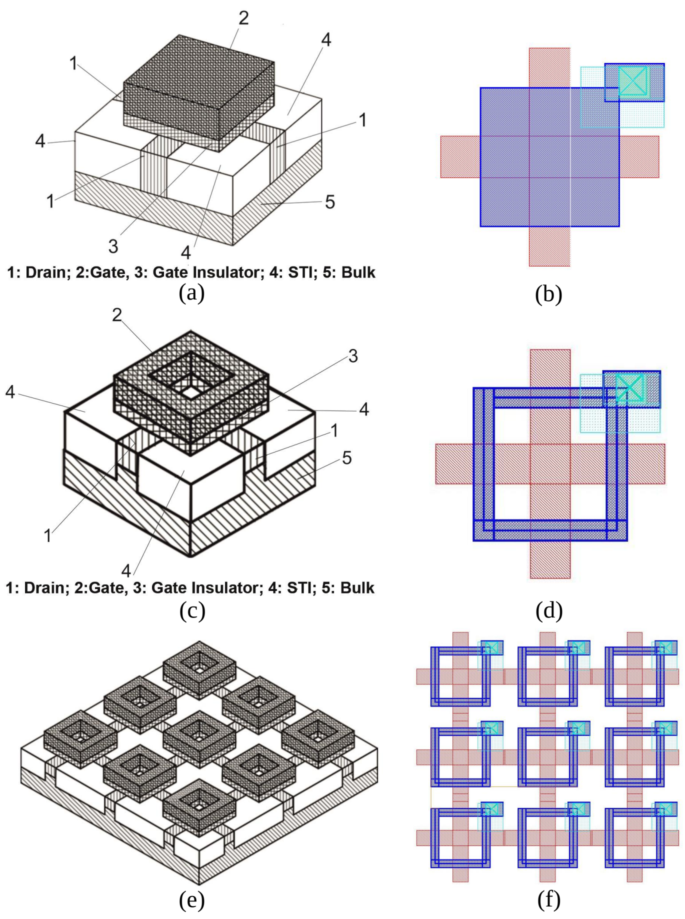
2
4
1
4
1
1
5
3
4
1: Drain; 2:Gate, 3: Gate Insulator; 4: STI; 5: Bulk
(a)
(b)
2
3
4
4
1
5
1
4
1: Drain; 2:Gate, 3: Gate Insulator; 4: STI; 5: Bulk
(c)
(d)
(e)
(f)

Fig. 3. Square-shaped four-terminal switch of [10]: (a) 3D view, and (b) layout; the proposed four-terminal switch: (c) 3D view, and (d) layout; the proposed 3×3 switching lattice: (e) 3D view, and (f) layout.

in TSMC 65nm process as an n-type switch, respectively. In our design, four independent transistors sharing a single source region are controlled by a single gate terminal, so it can be simulated using four NMOS transistors. Additionally, channel length and width of the device can be set independently since it has four distinct channels which are turned on and off simultaneously. Therefore, the channel length can be set to the minimum value just like a regular transistor. When carriers flow between any two terminals, they flow through two transistor channels serially; the effective channel length is 120nm that results in a width to length ratio of 120nm/120nm. Having a similar area, the proposed minimum size device offers nearly three times more conductivity compared to the minimum size square-shaped switch. Fig. 3(e) presents the 3D view of a 3×3 lattice structure formed with the proposed switches in a standard CMOS process; a layout of the lattice in TSMC 65nm process is shown in Fig. 3(f). Since metal contacts are not used for connecting switches, the lattices are very compact. The only metal connections in the switch structure are the gate contacts placed on the upper right corner of the gates. Here, the minimum switch pitch in a lattice is limited by the gate distance rules. The gate contact increases the minimum switch size. Of course, one can increase the size for more conductivity. The formula for the switch size is $(W+560\text{nm}) \times (W+510\text{nm})$; W is the transistor channel width.

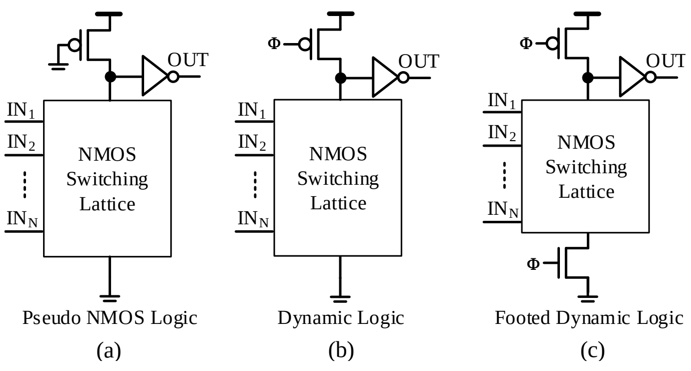

Fig. 4. (a) pseudo NMOS, (b) dynamic, and (c) footed dynamic circuit structures, all having NMOS switching lattices.

TABLE I
DETAILS OF THE LOGIC FUNCTIONS AND SUMMARY OF RESULTS OF LOGIC SYNTHESIS ALGORITHMS.

Instance	Function Details	*cadence_synth*	*abc_synth*	JANUS [8]	
	# of inputs - # of products - degree	# of two-terminal switches	# of two-terminal switches	size	# of four-terminal switches
5xp1_1	7 - 11 - 5	70	60	4x6	24
apex4_17	9 - 12 -	120	114	7x7	49
apex4_18	9 - 14 - 8	148	116	7x8	56
b12_02	8 - 7 - 5	58	28	4x4	16
ex5_07	8 - 10 - 4	66	50	3x8	24
ex5_10	6 - 7 - 3	30	26	3x6	18
ex5_12	8 - 9 - 3	44	32	3x5	15
ex5_21	8 - 10 - 3	56	52	3x7	21
misex1_02	7 - 5 - 5	48	40	5x4	20
mp2d_03	10 - 5 - 8	56	44	4x6	24
mp2d_04	10 - 6 - 9	58	40	7x3	21
sao2_01	10 - 20 - 10	152	136	12x7	84

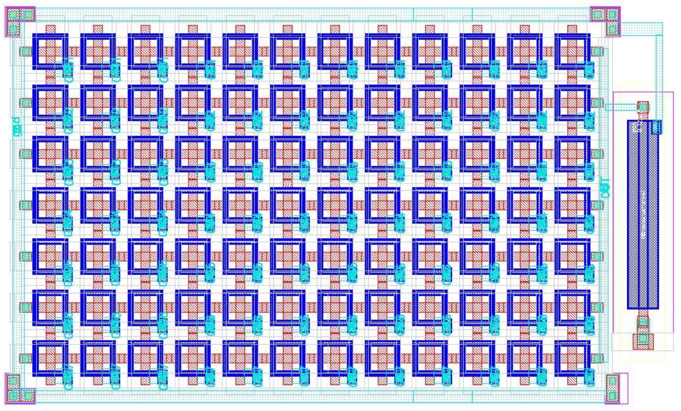

Fig. 5. Rotated layout of the switching lattice circuit using the pseudo NMOS logic for the function *sao2_01*. Layout area is nearly 8.7μm×5.4μm.

III. REALIZATION OF LOGIC FUNCTIONS USING SWITCHING LATTICES

We use a single lattice of NMOS switches as a pull-down network of a pseudo NMOS logic, dynamic logic, and footed dynamic logic [11]. Fig. 4 shows their circuit structures. In this figure, the inputs to the switching lattices are actually the literals of the logic function.

Although the pseudo NMOS logic implementation given in Fig. 4(a) is a simple and straightforward solution, we note that the difference between the best-case and the worst-case equivalent pull-down resistances can be very large in the designs using the pseudo NMOS logic. Since the pull-up network is a single PMOS transistor, which is always on, the PMOS transistor needs to be weaker than the worst-case pull-down configuration. Therefore, the low to high propagation delay ends up being significantly worse than the high to low propagation delay. If the pull-up transistor is too strong, the output low voltage level will be too high.

To tackle this propagation delay problem, a dynamic logic implementation is a viable option. In this case, the pull-up transistor is pulsed with a clock Φ as shown in Fig. 4(b). The output is evaluated only when the PMOS transistor is off. Therefore, the output voltage during the precharge is not a concern. The PMOS transistor in a dynamic logic circuit can be significantly larger than that in the pseudo NMOS circuit. Therefore, the low to high propagation delay during the precharge can be faster.

IV. EXPERIMENTAL RESULTS

We present implementations of 12 logic benchmark functions taken from [12] using two-terminal and four-terminal switches, all implemented in Cadence environment using TSMC 65nm CMOS technology. For the implementations, three main steps are followed that are logic synthesis, circuit design, and layout generation. Then post layout simulations are performed to obtain the results.

In the first step, we use two different logic synthesis methods for the implementations with two-terminal switches. In the first one, we directly give the truth tables of logic functions to the Cadence RTL compiler to perform logic optimization. This method is called *cadence_synth*. Also to further decrease the number of transistors with an aim of achieving better layout areas at the end, we use an optimization as i) apply a state-of-art logic synthesis tool on the logic function using a synthesis script; ii) map the design into the gates of a given library where the cost value of a gate is defined as the number of MOS transistors required to build the gate; iii) compute the design complexity in terms of the number of MOS transistors in the design; and iv) repeat this process for a number of synthesis scripts and keep the design with the least complexity. Here, we use ABC [13] as a logic synthesis tool, a total of 17 synthesis scripts, and the extended version of the *mcnc.genlib* library. This method is called *abc_synth*. For the implementations with four-terminal switches, the first step is achieved by JANUS [8].

Table I gives results for the used logic synthesis methods of *abc_synth*, *cadence_synth*, and JANUS with the details of the used benchmark functions. Here, *inputs*, *products*, and *degree* stand for the number of inputs, number of products, and the maximum number of literals in the products of the logic function in optimized SOP form, respectively. Examining the results in Table I, we see that the number of two-terminal switches found using *abc_synth* is always smaller than those obtained using *cadence_synth*. We also see that the solution of JANUS on each logic function offers much smaller number of switches compared to *abc_synth*.

In the second and third steps for the implementations with two-terminal switches, circuits are designed with the Cadence Genus tool using Cadence's digital library, and place and route is applied with the Cadence Innovus tool, respectively. On the

TABLE II
SUMMARY OF RESULTS OF DESIGNS IMPLEMENTED USING TWO-AND FOUR-TERMINAL SWITCHES WITH 65NM CMOS PROCESS.

Instance	Implementations with Two-Terminal Switches						Implementations with Four-Terminal Switches								
	CMOS - Cadence			CMOS - ABC+Cadence			Pseudo NMOS			Dynamic NMOS			Footed Dynamic NMOS		
	area	delay	power	area	delay	power	area	delay	power	area	delay	power	area	delay	power
5xp1_1	47.24	0.24	0.74	40.10	0.21	0.67	17.62	1. 64	4.07	17.62	0.22	4.25	18.85	0.54	0.27
apex4_17	81.31	0.34	1.04	79.17	0.29	1.06	30.20	5.69	2.24	30.20	0.49	2.20	31.74	1.04	0.10
apex4_18	98.93	0.33	1.33	78.07	0.25	1.09	34.09	5.19	2.27	34.09	0.52	2.44	35.75	1.44	0.50
b12_02	31.68	0.18	0.58	18.81	0.16	0.35	12.41	0.98	4.03	12.41	0.11	3.42	13.39	0.46	0.20
ex5_07	45.18	0.31	0.65	40.96	0.17	0.37	19.07	1.29	5.48	19.07	0.17	8.12	20.48	0.34	0.30
ex5_10	20.64	0.19	0.27	18.39	0.15	0.28	14.72	0.62	5.81	14.72	0.16	8.09	15.89	0.29	0.28
ex5_12	22.50	0.13	0.32	22.73	0.12	0.30	12.54	1.17	4.81	12.54	0.17	6.65	13.59	0.36	0.20
ex5_21	38.07	0.24	0.59	35.37	0.18	0.45	16.90	1.02	5.08	16.90	0.18	6.04	18.18	0.26	0.25
misex1_02	22.27	0.20	0.48	24.58	0.15	0.38	14.45	0.70	4.14	14.45	0.14	2.77	15.50	0.52	0.25
mp2d_03	28.93	0.20	0.4	30.69	0.16	0.48	17.62	3.02	2.66	17.62	0.26	3.33	18.85	0.91	0.24
mp2d_04	26.99	0.19	0.39	28.86	0.21	0. 38	14.64	1.87	2.55	14.64	0.11	1.66	15.70	1.15	0.23
sao2_01	94.10	0.42	1.07	95.95	0.27	1.41	46.83	9.88	1.60	46.83	0.14	1.17	48.70	2.66	0.51
Average	46.49	0.25	0.65	42.81	0.19	0.60	20.92	2.75	3.73	20.92	0,22	4,18	22.22	0.83	0.28

other hand for the switching lattices these steps are mostly manually done with the help of the Cadence Virtuso tool. While generating the layouts of switching lattices, the place step is done manually with ease due to the simple and regular structure of lattices. However, the route or the wiring step can be very complicated depending on the function to be implemented. In our case, for all benchmark functions, whole metal routing can be fit on the lattice and no peripheral area is wasted for the metal routing. All internal connections of the lattice can be drawn by the automated routing tool of Cadence using three metal layers. As an example, Fig. 5 shows the layout of the logic function *sao2_01*.

Table II presents layout area (in μm^2), worst-case delay (in ns), and average power dissipation (in µW) results of designs of logic functions given in Table I. To implement two-terminal switch based implementations, we consider two approaches differing by their logic synthesis tools, namely “Cadence” using *cadence_synth* for the first step and the mentioned Cadence tools for the second and third steps, and “ABC+Cadence” using *abc_synth* for the first step and the same Cadence tools for the second and third steps. To implement switching lattices, we consider three different logics, namely pseudo NMOS logic, dynamic logic and footed dynamic logic, explained in Section III. In order to compute delay and power dissipations, in post layout simulations, we use a test-bench including 10000 input patterns which are applied at 20Mhz.

Comparing two-terminal switch based implementations in Table II, we see that “ABC+Cadence” gives better results for all performance metrics in average, compared to “ABC+Cadence”. On the other hand, comparing four-terminals switched based ones, the best ones in average are “Pseudo NMOS”, “Dynamic NMOS”, and “Footed Dynamic NMOS” for area, delay, and power consumption, respectively. Also, wee see that the designs using four-terminal switches have nearly 2X smaller area in average than those using two-terminal switches. Area efficiency is more prominent in functions having a large number of inputs and products.

V. DISCUSSION AND FUTURE DIRECTION

In this paper, we show implementations of switching lattices to synthesize logic functions using the pseudo NMOS and dynamic logic structures under 65nm CMOS design process. Our performance improvement in terms of area, delay, and power consumption over conventional CMOS circuits is expected to get better if smaller CMOS technologies are used. The reason is that the smaller the CMOS technology the larger the interconnection problem and this favors switching lattices having dense, regular, and metal-connection-free structures. Here, there is also an important challenge. Layout regulations of most of the under 30nm technologies including the FinFET technologies only allows one dimensional alignment of transistors, meaning that transistor channels should be parallel to each other. Therefore new device geometries and layout techniques should be developed as future work.

REFERENCES

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

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

[3] M. C. Morgul and M. Altun, “Synthesis and optimization of switching nanoarrays,” in *2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*. IEEE, 2015, pp. 161–164.

[4] 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.

[5] 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.

[6] ——, “Composition of switching lattices for regular and for decomposed functions,” *MICPRO*, vol. 60, pp. 207–218, 2018.

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

[8] L. Aksoy and M. Altun, “A satisfiability-based approximate algorithm for logic synthesis using switching lattices,” in *DATE*, 2019.

[9] A. Bernasconi, F. Luccio, L. Pagli, and D. Rucci, “Literal selection in switching lattice design,” in *Advanced Boolean Techniques*. Springer, 2020, pp. 159–175.

[10] 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.

[11] N. Weste and D. Harris, *Integrated circuit design: International version: A circuits and systems perspective*. Pearson Education, 2010.

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

[13] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. [Online]. Available: https://people.eecs.berkeley.edu/ alanmi/abc/