Logic Synthesis for Emerging Technologies

Augusto Neutzling[1] and Renato P. Ribas[2]

[1] Cadence Design Systems, Inc., Cambridge, UK
[2] Institute of Informatics, UFRGS, Porto Alegre, Brazil
e-mail: auguston@cadence.com

***Abstract*— Emerging technologies are being considered to replace the conventional CMOS-based design that seems arriving to its end of life due to the limits of MOS transistor shrinking. However, since those novel devices are not necessarily switch-based ones, the traditional AND/OR logic synthesis process in the digital integrated circuit design flow tends to become inefficient, whereas threshold logic paradigm seems to be more appropriate for them. In this context, different methods for threshold logic synthesis, suitable for emerging technologies, are reviewed in this paper. The majority logic based design is also discussed herein since it represents a subset of threshold logic domain, and many new technologies have presented the 3-input majority Boolean function as the most basic logic gate. Experimental data, presented in previous works, are used to illustrate and compare the performance of the state-of-the-art logic synthesis methods related to.**

***Index Terms*— Threshold Logic; Majority Gate; Digital Circuit; Logic Synthesis; Emerging Technology.**

I. Introduction

Continuous scaling of MOSFETs has been the principal strategy for improving integrated circuit performance. However, as MOS transistor dimensions shrink, manufacturing imperfections and quantum effects become more critical and threaten to stop the CMOS scaling [1]. As a result, much effort has been done in order to develop new devices that may allow further progress in computation capability. Among these emerging technologies are carbon nanotubes and related graphene structures [2], single electron transistors (SET) [3], nanowire transistors [4], quantum-dot cellular automata (QCA) [5], nanomagnetic logic (NML) [6], resonant tunneling diodes (RTD) [7], spintronic-based devices [8], memristors or memristive devices [9], multiple independent-gate field effect transistors (MIGFETs) [10], and many other possibilities.

For these emerging technologies be successful, it has to present some characteristics that represent advancement and improvement over traditional MOS transistors [11]. These can be related to higher frequency operation, lower power consumption, smaller area, and so on. Additionally, it is welcome that the knowledge used in designing and fabricating CMOS circuits can be somehow adapted to the new technology to make the conversion process easier. In this sense, the logic synthesis process plays a crucial role in the digital circuit design, taking into account that many of those new technologies do not provide switch-based devices, becoming inefficient and even incompatible the conventional switch-based design circuitry based on AND/OR synthesis.

Particular design synthesis effort has been presented for specific technologies as observed for MIGFETS [12], spin-diode [13] and memresistive IMPLY [14]. On the other hand, many of these emerging devices have been demonstrated to be more appropriate for threshold logic design paradigm, like RDT, SET, QCA, NML and memristors. As a consequence, algorithms addressing threshold logic synthesis have been presented in the literature [15, 16, 17, 18, 19, 20, 21].

Moreover, some of emerging technologies are better suited for majority (MAJ) logic paradigm which can be considered as a subset of threshold logic domain [11][22]. Therefore, new EDA algorithms have been proposed in order to improve digital circuit design by exploiting MAJ gates [22, 23, 24, 25, 26, 27]. Such a change from AND/OR logic paradigm to the threshold logic one, in particular majority-based logic, impacts significantly the circuit synthesis process.

This paper presents an extended survey about threshold logic and MAJ-based logic synthesis, pointing out the main contributions of each work and the challenges for future improvement. Some experimental results already published in other papers have been shown and discussed herein in order to provide to the reader a better comprehension of the performance obtained on related work. In the organization of this paper, initially the more general threshold logic synthesis is reviewed followed by the discussion about the more specific MAJ-based synthesis.

II. Threshold Logic Design

This section presents the fundamentals about threshold logic field for a better understanding of this survey.

A. Terms and Definitions

A *threshold logic function* (TLF) is a Boolean function satisfying the following condition [28]:

$$f = \begin{cases} 1, & \textit{if} \sum_{i=1}^{n} x_i w_i \geq T \\ 0, & \textit{otherwise} \end{cases} \quad (1)$$

where x_i represents each Boolean input value $\{0, 1\}$, w_i is the weight of each input, and T is the function threshold value. Therefore, each input has a specific weight and the function has a threshold value. If the sum of active input weights (*i.e.*, inputs are equal to 1) is greater or equal than the threshold value, then the function evaluates to 1. Otherwise, the function evaluates to 0.

A TLF is completely represented by a compact vector [w_1,w_2,...,w_n;T], where w_1,w_2,...,w_n are the input weights and T is the function threshold value. A TLF has also been called 'linearly separable' function.

Digital Object Identifier 10.29292/jics.v16i1.484

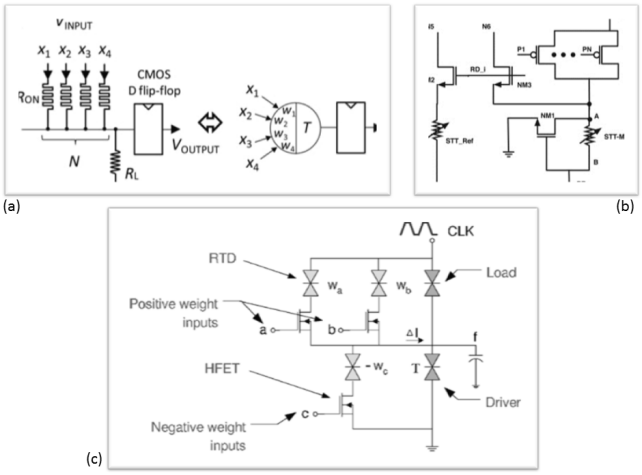

Fig. 1: Threshold logic gates based on: (a) memristor [33], (b) spintronic device [34], (c) RTD [15].

Although some complex functions are TLFs, there exist some simple functions that are not TLF. For instance, the function $h = x_1x_2 + x_3x_4$ cannot be represented in terms of input weights and threshold value. The *threshold logic identification* process verifies if a given Boolean function is TLF (or not) and compute the input weights and the corresponding threshold value. Many TLF identification algorithms are based on integer-linear-programming (ILP) [15, 16, 16, 17, 18, 29]. In [30], on the other hand, a complete system of inequalities is built using a similar strategy to ILP inequalities generation algorithms. However, unlike ILP-based approaches, the inequalities system is not solved. Instead, the algorithm speeds up the process by selecting some of the inequalities as constraints to the associated variables and computing the variable (input) weights in a bottom-up strategy. After this assignment, the consistency of the entire system is verified in order to check whether the weights have been correctly computed. Such a strategy has resulted in fast runtime and good quality of results (QoR). In a more recent work, in [31], the authors proposes a new necessary condition and the corresponding speedup strategies to the threshold function identification problem, claiming to obtain TLFs not found by the method presented in [30].

In the context of technology mapping, more specifically structural cuts, *threshold cut* is defined as a cut that the corresponding Boolean function is TLF.

Threshold logic gate (TLG), in turn, is a single primitive or a non-decomposable circuit that physically embodies the behavior expressed in Equation (1). TLGs can represent the implementation of complex functions. For instance, the TLG defined by $[4, 3, 3, 1, 1; 7]$ corresponds to $f = x_1x_2 + x_1x_3 + x_2x_3x_4 + x_2x_3x_5$. It is worth to notice that using larger threshold functions has the potential benefit of reducing the total number of gates needed to implement digital circuits. Several topologies of TLGs have been proposed for CMOS and emerging nanotechnologies. A survey with more than 50 TLG circuitries are presented by Beiu *et. al*, in [32]. Moreover, TLG designs based on memristor [33], spintronic device [34] and RTD [15] technologies have also been proposed, as illustrated in Fig. 1.

Threshold logic network (TLN), on the other hand, is a netlist of TLFs and interconnections. A TLN can be implemented using TLGs, since each TLF can be directly implemented through a single TLG. The total area of TLN corresponds to the sum of the TLG areas. Notice that the TLG area depends directly on the technology adopted to build such a gate. However, since no threshold-oriented technology currently available is mature enough to fabricate reliable TLGs in large scale, synthesis tools usually consider that each threshold gate presents the same area. Therefore, the total circuit area is commonly estimated through the overall number of TLGs instantiated in the mapped circuit.

Although it is hard to define a general TLG area estimation, some of them are more suitable to particular technologies. For instance, when designing a TLG by using RTDs, each input weight and function threshold values determine the diode physical area. Therefore, the gate area is directly related to the sum of the weights and the threshold value, as follows [15]:

$$A_{\text{TLG}} = T + A_u(\sum_{i=1}^{k} w_i) \tag{2}$$

where k is the number of TLG inputs (fanin), w_i is the weight of input i, A_u is the unit area of an RTD with w=1, and T is the threshold of the gate.

In other technologies, such as memristors [9][33] and spintronic-based devices [34][35], the input weight is set by applying a voltage over the device during a moment, so not impacting the device physical dimensions. In these cases, each input is associated to a single device (with the same area) and, as a consequence, the most appropriate gate area estimation metric is the number of gate inputs.

B. Threshold Logic Synthesis

The goal of threshold logic synthesis is to generate a TLN where each TLF can be directly implemented through a single TLG. The design flow adopted by many works starts by performing a LUT-based technology mapping [15, 16, 17, 18, 36]. This first mapping task results in a netlist of Boolean functions with restricted fanin. Afterwards, these methods identify which Boolean functions are TLF and then include them into the final solution. For each non-TLF, they generate a sub-network. The main differences among these approaches are: (i) the procedure of the threshold logic identification, and (ii) the generation of the sub-TLN from a non-threshold function.

Zhang *et al.*, in [15], and Subirats *et al.*, in [16], use ILP to perform the TLF identification. For non-TLFs, the Zhang's method decomposes the function into AND/OR sub-fuctions, which are always TLFs, and select nodes through a heuristic way in order to combine them, checking whether the resulting function is TLF. The Subirats' approach, on the other hand, is based on the function truth table description. It selects recursively a variable and performs the Shannon decomposition up to find TLF sub-functions. The Subirats' method improves the Zhang's results in terms of the number of gates and circuit logic depth. However, the Subirats' approach produces only two-level TLNs without fanin restrictions, being more suitable to neural network design.

In [17], Gowda *et al.* propose a heuristic approach to identify TLF. They adopt both binary decision diagrams (BDD) and a factorized tree structure (called max literal factor tree - MLFT) in order to generate a TLN. The method breaks recursively the initial expression tree into sub-expressions, identifying sub-trees that represent TLF and assigning input weights. The method proposed by Palaniswamy *et al.*, in [18], improves the Gowda's approach [17]. It looks for circuit outputs that can be implemented as a single TLF. Both the Gowda's and the Palaniswamy's methods suffer from execution time as the main bottleneck, and the solutions depend strongly on the initial structure (BDD or MLFT), in particular, on the ordering of tree nodes.

In [36], it is proposed a method based on a TLG association process through the principle called functional composition, which is based on dynamic programming [37]. The algorithm associates simpler sub-solutions with known design costs, *e.g.* the number of gates, in order to produce the final solution with minimum cost. In order to identify whether the Boolean function created from such an association is TLF, the method adopts the heuristic method proposed in [38]. This approach presents improved results in terms of TLG count when compared to previous approaches. However, the design optimization in respect to the circuit logic depth is not so significant. Moreover, the execution time is also a limitation and the approach does not scales for TLFs with more than six variables (inputs).

The threshold logic synthesis methods mentioned above generate a TLF network from a general Boolean function netlist. Another set of approaches focuses in the optimization starting from a TLN. Methods for TLG-based circuit rewiring are presented by Kuo *et al.*, in [39], and by Lin *et al.*, in [20]. Kuo's approach focuses only on circuit restructuring for satisfying a new fanin constraint and does not take into account the area and logic depth minimization issue. Lin's approach, on the other hand, represents a heuristic for rewiring the circuit by minimizing the summation of input weights and threshold value. In [21], Chen *et al.* propose an analytical approach based on collapsing two threshold gates in order to minimize the total number of TLGs. Annampedu's method, in [40], receives as input a given (already identified) single-output threshold function. Therefore, this method is not able to treat a general logic circuit with multiple outputs corresponding to Boolean functions not necessarily identified as threshold ones. The main goal of Annampedu's method is to restrict the fanin of a given threshold function implementation. Furthermore, Kulkarni *et al.*, in [41], propose TLG-based approaches to reduce the circuit area and power consumption without loss of performance. However, the technology mapping is performed by a commercial tool using conventional (*i.e.*, non-threshold based) standard cell library. This method replaces some standard flip-flops by threshold logic sequential cells [41]. As a consequence, a hybrid netlist comprising both TLGs and conventional logic gates is provided.

Notice that all of mentioned threshold logic synthesis approaches focus basically on synthesizing single-output non-TLF. The first step of previous methods relies on a complete synthesis process which disregards threshold logic domain. Therefore, they do not explore the entire circuit where they would find, for instance, TLFs between different functions in the netlist. Such a bootleneck has been overcome in [42] [19], where Neutzling *et al.* propose a logic synthesis flow that identifies TLFs before the circuit covering task. It is based on a three-stage procedure, as depicted in Fig. 2(b): (*i*) a complete cut enumeration, storing Boolean functions of cuts in the design; (*ii*) the identification of TLFs related to this set of computed cuts; and (*iii*) the technology mapping considering thresholdness of pre-computed functions. By doing so, this approach is able to discard non-TLF cuts and provides the corresponding threshold network from the first covering action. Such a strategy allows the exploitation of multi-objective technology mapping algorithms.

Finally, in [43], the authors propose an optimization method for TLN based on observability don't-care-based node merging. To reduce gate count in a TLN, it iteratively merges two gates that are functionally equivalent or whose differences are never observed at the primary outputs. Furthermore, it is able to identify redundant wires and replace wires for removing more gates. Basically, the proposed method is primarily adapted from an ATPG-based node-merging approach which works for conventional Boolean logic networks. To extend the approach for TLN, a method for computing mandatory assignments of a stuck-at fault test on a threshold gate and another one for conducting logic implication in a TLN have been developed. Additionally, to achieve a better optimization quality, the proposed method has been integrated with other optimization methods. The same authors, in [44], have recently improved this work by proposing don't-care-based node minimization.

C. Comparing Threshold Logic Synthesis Approaches

Some previous threshold logic synthesis approaches have been compared through three experiments presented in [19]. The first one compares the number of TLGs and the circuit logic depth to the results obtained from both strategies presented by Chen *et al.*, in [21], and by adopting a commercial tool. In the second one, the Neutzling's approach is compared to the Gowda's method, in [17], and the Palaniswami's method, in [18], in terms of the number of TLGs. It has been done because the Chen's work does not compare itself to those approaches. In the third experiment, the circuit area results obtained by the Neutzling's method are compared to the ones presented in [15] and in [20] in terms of the sum of input weights and threshold value.

Since the Chen's method already provides improvement of 28% in TLG count and 14% in logic depth when compared to the Zhang's approach [15], it has been considered in the comparison to the Neutzling's approach, in [19]. It was carried out taking into account the IWLS 2005 benchmark suite [45]. Table I shows the obtained results in terms of TLG count and circuit logic depth. When limiting the TLGs to six inputs, the Neutzling's method reduced the TLG count in 94% of the circuits, with reductions up to 39% of such a count, being 20% on average. The logic depth has been reduced in all applied benchmark circuits, with reductions up to 64% and being 53% on average. The runtime is less than one second per circuit synthesis.

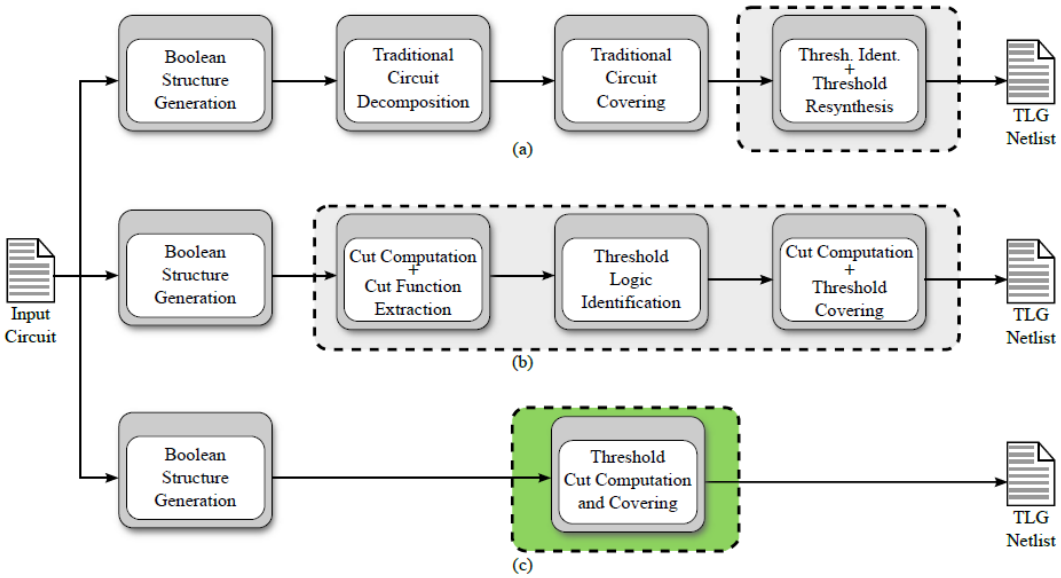

Fig. 2 Different threshold logic synthesis flows: (a) previous works from other authors, (b) proposed in [42]; (c) proposed in [19].

In order to exploit the gate level scalability, circuits have been synthesized taking into account TLGs with up to 15 inputs. In this case, the TLG count has been reduced in all circuits, with reductions up to 47% and being 25% on average. The reduction in terms of logic depth has been up to 67%, being 57% on average. In the same experiment, circuits have also synthesized by adopting a commercial tool. To do that, it has been provided the tool with a cell library composed by all NPN threshold functions with up to six variables. Notice that, although the commercial tool improves the Cheng's results in terms of TLG count, the second Neutzling's approach method has been able to improve these results even more. On average, the commercial tool improves 7% whereas the Neutzling's approach improves 15%. Moreover, TLG count and circuit logic depth are simultaneously reduced by the proposed flow, whereas the commercial tool increases the Cheng's results in around 38% in terms of logic depth.

In [18], the authors present two different improvements, named BDD decomposition method (BDM) and ZDD decomposition method (ZDM), to the max literal factor tree (MLFT) method proposed by Gowda *et al.*, in [17]. The results shown in Fig. 3 present the TLG count obtained by these methods and the Neutzling's one. The ISCAS'85 set of benchmarks has been applied for this evaluation. When compared to the MLFT approach, BDM and ZDM methods provide an average TLG count reduction of 12% and 17%, respectively. The average reduction obtained the Neutzling's method is about 65% and 48% when compared to MLFT and ZDM, respectively.

Finally, in [19], the Neutzling's approach has been compared to the work presented by Lin *et al.*, in [20], in terms of the summation of input weights and threshold value. The Lin's method starts from a TLG netlist (*i.e.*, a given TLN) generated by the Zhang's method, in [15], and performs a rewiring procedure, optimizing the TLG area cost function. Table **??** shows the results from this experiment. In [20], the Lin's method improves the Zhang's results for all benchmarks, obtaining a reduction of 4% on average. The Neutzling's approach does not depend on a preliminary threshold synthesis and optimizes the cost function performing a threshold logic technology mapping directly over the original circuit description. Therefore, the Zhang's results have been improved for all benchmarks, so reducing the circuit area up to 46%, being 31% on average.

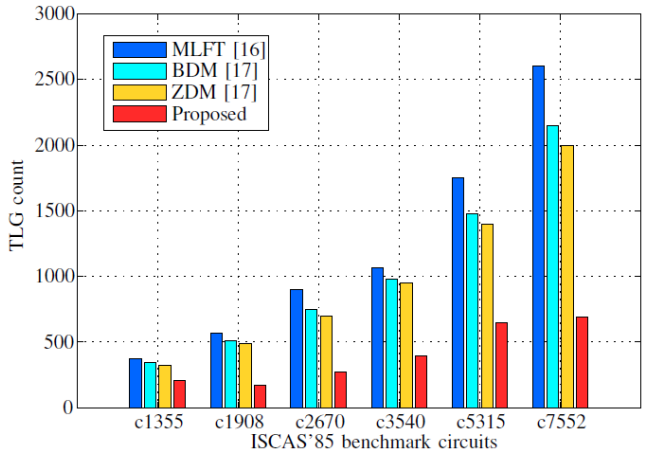

Fig. 3: Comparison between the Palaniswamy's, Gowda's and Neutzling's methods presented in [19].

The Neutzling's threshold logic synthesis flow explores the LUT-based technology mapping strategy, which allows for near-optimum circuit logic depth covering [46]. From such a covering, three different mapping goals can be targeted in circuit area optimization. The first one chooses a cut that decreases the area even when increasing the logic depth (area oriented). The second one never replaces a cut if the logic depth is increased (delay oriented). Finally, an intermediate strategy chooses a cut that decreases the area whether the increasing in logic depth is less than a given predefined percentage (relaxed delay). In the experiments, it has allowed to increase the delay up to 30%. A comprehensive set of experimental results when addressing the mapper proposed in [19] to the aforementioned goals and targeting different threshold-based circuit area estimations is presented.

Notice that two ternary possibilities have been varied in this experiment: the mapping goal, which can be area oriented, delay oriented or relaxed one; and the area estimations that can be considered as the number of TLGs, the summation of input weights and threshold values, or the overall gate fanin. This yields nine different solutions to each synthesized circuit.

In [19], the results obtained by the Neutzling's approach are presented in terms of circuit area and logic depth taking into account different benchmark suites, such as the EPFL more-than-million (MTM), arithmetic and random-control [47], ISCAS'85 [45] and the opencore circuits [48]. The circuit level scalability has been verified by synthesizing benchmarks comprising more than 20 million AIG nodes.

Table I.: Comparison of TLG count between the Chen's and the Neutzling's approaches, and by applying a commercial tool, using the Chen's one as reference [19].

Circuit	*Chen* [21]	*EDA*	*K=6* [19]	*K=15* [19]
spi	1614	(0.78)	(0.74)	(0.71)
systemcaes	5333	(0.82)	(0.79)	(0.77)
tv80	3559	(0.91)	(0.81)	(0.76)
ac97_ctrl	6194	(1.01)	(0.95)	(0.91)
sasc	333	(1.04)	(0.92)	(0.89)
usb_funct	6842	(0.93)	(0.82)	(0.78)
mem_ctrl	4721	(0.77)	(0.76)	(0.71)
systemcdes	1377	(0.90)	(0.82)	(0.77)
i2c	482	(0.87)	(0.76)	(0.69)
pci_bridge32	10497	(0.91)	(0.89)	(0.87)
aes_core	10057	(0.97)	(0.89)	(0.78)
simple_spi	436	(1.01)	(0.85)	(0.82)
des_area	2011	(1.01)	(1.01)	(0.95)
wb_conmax	21956	(0.87)	(0.82)	(0.80)
pci_spoci_ctrl	399	(0.64)	(0.61)	(0.53)

Table II.: Comparison of circuit logic depth between the Chen's and the Neutzling's approaches, and by applying a commercial tool, using the Chen's one as reference [19].

Circuit	*Chen* [21]	*EDA*	*K=6* [19]	*K=15* [19]
spi	19	(1.21)	(0.53)	(0.47)
systemcaes	33	(0.88)	(0.42)	(0.42)
tv80	30	(1.40)	(0.47)	(0.37)
ac97_ctrl	7	(1.43)	(0.57)	(0.43)
sasc	7	(1.14)	(0.43)	(0.43)
usb_funct	19	(1.42)	(0.53)	(0.47)
mem_ctrl	23	(1.30)	(0.39)	(0.35)
systemcdes	19	(1.16)	(0.47)	(0.47)
pci_bridge32	21	(1.95)	(0.38)	(0.33)
aes_core	17	(1.29)	(0.53)	(0.53)
simple_spi	8	(1.38)	(0.50)	(0.38)
des_area	20	(1.40)	(0.55)	(0.50)
wb_conmax	13	(1.62)	(0.62)	(0.54)
pci_spoci_ctrl	12	(1.50)	(0.42)	(0.33)

III. MAJORITY-BASED LOGIC DESIGN

Some emerging nanotechnologies are well suited for digital integrated circuit design based on majority logic [11][22]. The most prominent case is the quantum-dot cellular automata (QCA) technology. The basic gate in QCA technology is the MAJ-3 one, as illustrated in Fig. 4(a). Nevertheless, majority gates with fanin larger than three have also been proposed [49, 50]. In QCA circuitry, larger majority gates can be obtained by using a staircase pattern, as depicted in Fig. 4(b). It is still unclear the maximum size of a QCA-based majority gates using such a structure. Spin-based devices can also potentially implement high fanin majority gates [51][8]. However, the area overhead when high fanin majority gates are considered must be carefully evaluated.

Moreover, it is known that the class of functions that can be implemented by majority gate with unbounded fanin is equivalent to the class of threshold logic functions [28]. This equivalency allows to extend the threshold logic synthesis flow to perform majority logic synthesis while taking into account the impact of fanin on the gate area.

A. Terms and Definitions

An n-input majority function (MAJ-n), where n is odd, can be seen as a special case of TLF where each input weight is equal to 1 and the threshold value T is given by [28]:

$$T = (n+1)/2 \tag{3}$$

For instance, MAJ-3 and MAJ-5 functions are equivalent to the following TLFs $[1,1,1;2]$ and $[1,1,1,1,1;3]$, respectively.

B. Majority Logic Synthesis

Logic synthesis methods that focus on majority logic aim to obtain an optimized network of majority gates. The most adopted design flow strategy can be summarized into two steps [22, 23, 24, 25, 26]: (*i*) to perform an FPGA-based technology mapping over the look-up table (LUT) structure, and (*ii*) to decompose each LUT into a network of majority gates. Notice that during the second step a single LUT may require several gates to be implemented. Therefore, minimizing the number of LUTs does not necessarily lead to the minimal number of majority gates. Alternatively, in [52] and in [53], the authors propose the majority-inverter graph (MIG) structure, which is based on MAJ-3 logic, as an efficient way to improve the digital circuit design. However, it is not clear how majority gates with more than three inputs can be effectively exploited in such a MIG-based synthesis.

On the other hand, since there are many efforts to build majority gates with more then three inputs [49, 50, 51, 8],

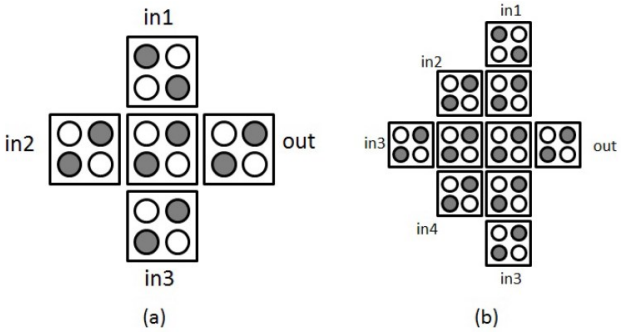

Fig. 4 QCA majority gates: (a) MAJ-3 and (b) MAJ-5 [10].

the development of logic synthesis methods for MAJ-n logic becomes necessary [26]. In [19], an effective technology mapping for maj-n logic is proposed. In such approach, after an FPGA-based technology mapping, each LUT is converted to a majority gate such that no further decomposition is required.

C. Comparison of MAJ Synthesis Approaches

A set of experiments presented in [27] compares the proposed method in such a reference to the approaches proposed by Wang *et al.*, in [25], by Amarù *et al.*, in [52] and by Soeken et al., in [53] for MAJ-3 synthesis.

Moreover, it has also been explored the use of MAJ-5 not addressed by previous works. The MAJ-3 and MAJ-5 gates are assumed to have the same area. The main idea of this analysis is to simply illustrate possible gains obtained through majority gates with more inputs. However, such improvements are only possible when the gate area does not increase too rapidly with the number of inputs.

In Table IV, columns 2 and 3 present, respectively, the MAJ-3 gate count and the circuit logic depth obtained by Wang's method [25]. Columns 4 and 5 show the results from Neutzling's method [27] when restricting the synthesis to MAJ-3 logic. The improvement with respect to the number of gates and logic depth have been up to 36% (being 16% on average) and up to 32% (being 13% on average), respectively. In turn, columns 6 and 7 present the results taking into account also the MAJ-5 logic in the synthesis. In this case, the Neutzling's maj synthesis approach have reduced up to 55% the gate count (being 46% on average) and up to 53% the logic depth (being %26 on average).

The work presented by Amarù *et al.*, in [52], introduces the MIG data structure. Although such a structure does not originally target majority-based integrated circuits, it can be directly translated into a MAJ-3 logic network. Table V shows the comparison between both approaches, restricted to MAJ-3 and considering also MAJ-5. When only MAJ-3 logic is addressed, the Neutzling's MAJ method yields a lower gate count at cost of increased logic depth. On the other hand, when considering MAJ-5, the average number of gates has been improved more than 40% while improving also the average circuit logic depth in around 17%.

Finally, the work presented by Soeken *et al.*, in [53], proposes algorithms for exact synthesis of Boolean logic networks using satisfiability modulo theories (SMT) solvers over MIGs. Table VI shows an average reduction both in number of gates (14%) and logic depth (35%) when using maj gates up to 3 inputs. When allowing also MAJ-5 gates, the average improvements are 39% in terms of total number of gates and 56% in terms of logic depth.

Another set of experiments presented in [27] compares the Neutzling's method to the work described in [26]. Since the previous work can only handle MAJ-3 and MAJ-5 gates, the same restriction has been set to the Neutzling's MAJ synthesis approach. In this experiment, both MAJ-3 and MAJ-5 gates are assumed to have the same physical area. Thus, the number of gates becomes the metric for area. Notice that there are cases when such an assumption is valid. For instance, in the USE methodology in [54]. It is also important to notice that the synthesis process described in [26] does not directly consider the gate area. In contrast, the Neutzling's maj method uses the gate area during the synthesis process.

Table IV.: Comparison between Wang's and Neutzling's methods, based on MAJ-3, presented in [27].

	Wang's		MAJ-3 [27]		MAJ-5 [27]	
circuit	Gates (total)	Level (total)	Gates (ratio)	Level (ratio)	Gates (ratio)	Level (ratio)
alu2	329	18	0.98	1.44	0.62	1.22
c8	108	8	0.99	0.88	0.60	0.88
k2	1193	19	0.72	0.68	0.49	0.58
frg2	568	15	1.07	0.80	0.63	0.73
apex6	662	17	0.86	0.71	0.55	0.59
example2	241	9	0.92	1.00	0.63	0.89
vda	670	14	0.64	0.79	0.45	0.79
frg1	102	17	0.65	0.71	0.38	0.47
x1	253	11	0.88	0.82	0.53	0.55
ttt2	144	10	0.82	1.10	0.53	1.00

Table V.: Comparison between Amarù's and Neutzling's methods, based on MAJ-3, presented in [27].

	Amarù [52]		MAJ-3 [27]		MAJ-5 [27]	
circuit	Gates (total)	Level (total)	Gates (ratio)	Level (ratio)	Gates (ratio)	Level (ratio)
systemcdes	2453	19	0.91	1.16	0.66	0.74
spi	3337	19	0.78	1.37	0.49	0.84
mem_ctrl	7143	19	0.94	1.21	0.61	0.74
tv80	7397	30	0.82	1.30	0.52	0.77
systemcaes	9547	25	0.83	1.16	0.54	0.80
ac97_ctrl	10745	8	0.96	1.13	0.61	0.75
usb_funct	12995	19	0.90	1.32	0.56	1.00
aes_core	20947	18	0.88	1.28	0.63	0.78
DSP	40517	34	0.87	1.68	0.56	1.35
des_perf	67194	15	1.03	1.07	0.76	0.67

Table VI.: Comparison between Soeken's and Neutzling's methods, based on MAJ-3, presented in [27].

	Soeken [53]		MAJ-3 [27]		MAJ-5 [27]	
circuit	Gates (total)	Level (total)	Gates (ratio)	Level (ratio)	Gates (ratio)	Level (ratio)
adder	513	130	1.74	0.99	1.24	0.99
bar	3336	12	0.92	0.92	0.66	0.67
div	35993	3607	0.65	0.94	0.71	0.62
hyp	196842	11317	1.14	1.19	0.85	0.76
log2	32060	444	1.03	0.60	0.69	0.38
max	2479	276	1.04	0.82	0.71	0.43
multiplier	22634	165	1.10	1.16	0.75	0.79
sin	5416	255	0.98	0.48	0.70	0.32
sqrt	24170	6073	1.00	0.58	0.72	0.36
square	17562	130	0.91	1.29	0.68	1.00
arbiter	8957	63	1.32	0.59	0.70	0.70
cavlc	757	19	0.75	0.58	0.49	0.37
ctrl	139	10	0.68	0.50	0.52	0.30
dec	328	4	0.93	0.75	0.93	0.75
i2c	1329	23	0.66	0.39	0.44	0.26
int2float	263	18	0.70	0.56	0.46	0.33
mem_ctrl	45034	144	0.49	0.33	0.31	0.17
priority	993	245	0.38	0.48	0.25	0.24
router	220	54	0.62	0.28	0.38	0.15
voter	7767	67	1.06	0.75	0.97	0.52

Therefore, this method should be more effective when considering different area values for different gates. Table VII summarizes the results. Overall, there is an average reduction of 10% on the number of gates and 14% on the number of levels. Experiments also demonstrate that the obtained reductions can be even higher when allowing majority gates with more than 5 inputs to be also used for mapping.

Table VII.: Comparison between Mishra's and Neutzling's methods, based on MAJ-3 and MAJ-5, presented in [27].

	Mishra [26]			Neutzling [27]		
Circuit	Maj3	Maj5	Total	Maj3	Maj5	Total
b1	6	0	6	1	3	4
cht	42	39	81	1	81	82
example2	128	62	190	90	78	168
frg1	12	49	61	10	28	38
pcler8	32	17	49	15	29	44
tt2	29	54	83	38	47	85
unreg	48	18	66	33	32	65
k2	206	366	572	266	365	631
frg2	168	207	375	139	240	379

The use of majority gates with large fanin has been evaluated in [27]. In this case, it is important to observe that the gate area might increase with respect to the number of inputs. Therefore, reducing the number of gates does not necessarily reduce the final circuit area.

Once such an analysis is not targeting a specific technology, it is important to take into account different majority gate area relationship to the number of inputs since it can vary from one technology to another. Therefore, the gate area has been estimated according to the fanin n and a parameter α that establishes such an area relationship, as follows:

$$gateArea = (\frac{n}{3})^{\alpha} \quad (4)$$

Through this equation, when $\alpha = 0$, the same circuitry area is assumed for all MAJ-n gates, regardless of the number of inputs. On the other hand, when $\alpha = 1$, the gate area becomes directly related to the gate fanin. Notice that the area of MAJ-3 gate has been adopted as reference, being equal to 1.

Notice that Equation (4) is intended to be used as a first order estimation for the gate area. This equation can be particularly useful when searching for new designs of MAJ-n gates. In fact, more accurate estimations can be used when available. For instance, two possible area estimations for a MAJ-n gate in QCA technology could be the number of cells in the gate as well as the area of the minimum enclosing rectangle.

The same benchmark circuits shown in Table IV have been adopted in these experiments. In Fig. 5, each curve represents the total number of MAJ-3, MAJ-5, MAJ-7 and MAJ-9 instantiated into the mapped circuits when varying α from 0 to 1.4. For α greater than 1.4, the results are kept unchanged.

In terms of logic behavior, the most compact design of MAJ-5 comprises four MAJ-3 gates [55]. Therefore, the area of MAJ-5 could be as much as four times larger than the corresponding area of MAJ-3, leading to a maximum value of $\alpha = 2.7$. However, it has been observed that no MAJ-5

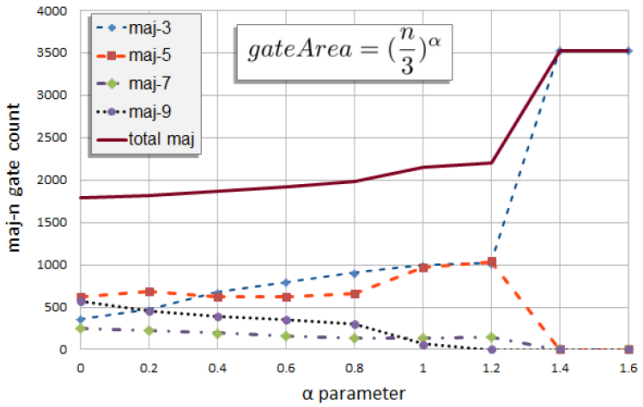

Fig. 5: Impact of fanin n in majority gate area estimation, given by Equation (4), by assigning different values of α parameter in the circuit synthesis.

has been instantiated when $\alpha \geq 1.4$. The main reason is the fact that the most MAJ-5 instances have been used to implement OA21 function $f = (x_0(x_1 + x_2))$ and AO21 function $f = (x_0 + (x_1.x_2))$, which can be built both by using only two MAJ-3 gates. As a result, the MAJ-5 area should be at most twice the MAJ-3 area to become useful, leading to a maximum value of $\alpha = 1.36$. Notice that, if the MAJ-n area is estimated by the number of QCA cels, then the MAJ-5 shown in Fig. 1(b) has exactly twice the MAJ-3 area, being in the boundary condition.

Some usual functions in digital circuit design, such as $f1 = (x_3.x_0(x_1 + x_2))$ and $f2 = (x_3 + x_0 + (x_1.x_2))$, can be built using a MAJ-9 gate but not a MAJ-7. It explains why MAJ-9 is more used than MAJ-7 for $\alpha \leq 0.8$. On the other hand, for $\alpha > 0.8$, using MAJ-3 and MAJ-7 to implement $f1$ and $f2$ leads to a smaller area than by considering only MAJ-9.

Finally, the synthesis considering majority gates with unbounded fanin and α equal to 0 have also been carried out. In such an evaluation, majority gates with up to 123 inputs have been taken into account in the final circuit. It indicates that majority gates with large fanin can become useful if the gate size is kept close to MAJ-3 gate area.

IV. Conclusions

This paper presented an extended survey about logic synthesis considering threshold logic and majority-based logic suitable for emerging nanotechnologies. It is clear that there is a room for future development in order to attend the design particularities of emerging technologies. It can be done by adapting existing traditional AND/OR synthesis (switch-based design) or by creating novel design environments and tools in the threshold logic domain.

References

[1] D. Frank, R. Dennard, E. Nowak, P. Solomon, Y. Taur, and H. Wong, "Device scaling limits of si mosfets and their application dependencies," *Proceedings of the IEEE*, vol. 89, no. 3, pp. 259–288, Mar. 2001.

[2] M. Schroter, M. Claus, P. Sakalas, M. Haferlach, and D. Wang, "Carbon nanotube fet technology for radio-frequency electronics: State-of-the-art overview," *IEEE Journal of Electron Devices*, vol. 1, no. 1, pp. 9–20, Jan. 2013.

[3] P. Koppinen, M. Stewart, and N. Zimmerman, "Fabrication and electrical characterization of fully cmos-compatible si single-electron devices," *IEEE Trans. on Electron Devices*, vol. 60, no. 1, pp. 78–83, Jan. 2013.

[4] K. Jansson, E. Lind, and L. Wernersson, "Performance evaluation of III–V nanowire transistors," *IEEE Trans. on Electron Devices*, vol. 59, no. 9, pp. 2375–2383, Sep. 2012.

[5] D. Tougaw and M. Khatun, "A scalable signal distribution network for quantum-dot cellular automata," *IEEE Trans. on Nanotechnology*, vol. 12, no. 2, pp. 215–224, Mar. 2013.

[6] N. Anderson and S. Bhanja, "Field-coupled nanocomputing," in *Springer*, 2014.

[7] H. Pettenghi, M. Avedillo, and J. Quintana, "Improved nanopipelined RTD adder using generalized threshold gates," *IEEE Trans. on Nanotechnology*, vol. 10, no. 1, pp. 155–162, Jan. 2011.

[8] S. Klingler, P. Pirro, T. Brächer, B. Leven, B. Hillebrands, and A. Chumak, "Design of a spin-wave majority gate employing mode selection," *Applied Physics Letters*, vol. 105, no. 15, p. 152410, 2014.

[9] A. Maan, D. Jayadevi, and A. James, "A survey of memristive threshold logic circuits," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 28, no. 8, pp. 1734–1746, Aug. 2017.

[10] L. Amarú, G. Hills, P. Gaillardon, S. Mitra, and G. De Micheli, "Multiple independent gate fets: How many gates do we need?" in *Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015.

[11] L. Amarú, P.-E. Gaillardon, S. Mitra, and G. De Micheli, "New logic synthesis as nanotechnology enabler," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2168–2195, Nov. 2015.

[12] J. Baqueta, F. Marranghello, V. Possani, A. Neutzling, A. Reis, and R. Ribas, "Binary adder circuit design using emerging migfet devices," in *Proc. of Int'l Symp. on Quality Electronic Design (ISQED)*, 2017, pp. 236–242.

[13] M. Martins, F. Marranghello, J. Friedman, A. Sahakian, R. Ribas, and A. Reis, "Spin diode network synthesis using functional composition," in *Proc. of Symp. on Integrated Circuits and Systems Design (SBCCI)*, 2013, pp. 236–242.

[14] F. Marranghello, V. Callegaro, A. Reis, and R. Ribas, "Four-level forms for memristive material implication logic," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1228–1232, May 2019.

[15] R. Zhang, P. Gupta, L. Zhong, and N. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 107–118, Jan. 2005.

[16] J. Subirats, J. Jerez, and L. Franco, "A new decomposition algorithm for threshold synthesis and generalization of Boolean functions," *IEEE Trans. on Circuits and Systems I*, vol. 55, no. 10, Nov. 2008.

[17] T. Gowda, S. Vrudhula, N. Kulkarni, and K. Berezowski, "Identification of threshold functions and synthesis of threshold networks," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, May 2011.

[18] A. Palaniswamy and S. Tragoudas, "Improved threshold logic synthesis using implicant-implicit algorithms," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, no. 3, May 2014.

[19] A. Neutzling, J. Matos, A. Mishchenko, A. Reis, and R. Ribas, "Effective logic synthesis for threshold logic circuit design," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, May 2019.

[20] C.-C. Lin, C.-Y. Wang, Y.-C. Chen, and C.-Y. Huang, "Rewiring for threshold logic circuit minimization," in *Proc. of the Conf. on Design, Automation & Test in Europe (DATE)*, 2014.

[21] Y.-C. Chen, R. Wang, and Y.-P. Chang, "Fast synthesis of threshold logic networks with optimization," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 486–491.

[22] R. Zhang, P. Gupta, and N. Jha, "Majority and minority network synthesis with application to QCA-, SET-, and TPL-based nanotechnologies," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 7, pp. 1233–1245, July 2007.

[23] R. Zhang, K. Walus, W. Wang, and G. Jullien, "A method of majority logic reduction for quantum cellular automata," *IEEE Trans. on Nanotechnology*, vol. 3, no. 4, pp. 443–450, Dec. 2004.

[24] K. Kong, Y. Shang, and R. Lu, "An optimized majority logic synthesis methodology for quantum-dot cellular automata," *IEEE Trans. on Nanotechnology*, vol. 9, no. 2, pp. 170–183, Mar. 2010.

[25] P. Wang, M. Niamat, S. Vemuru, M. Alam, and T. Killian, "Synthesis of majority/minority logic networks," *IEEE Trans. on Nanotechnology*, vol. 14, no. 3, pp. 473–483, May 2015.

[26] V. Mishra and H. Thapliyal, "Heuristic based majority/minority logic synthesis for emerging technologies," in *Int'l Conf. on VLSI Design and Int'l Conf. on Embedded Systems (VLSID)*, 2017, pp. 295–300.

[27] A. Neutzling, F. Marranghello, J. Matos, A. Reis, and R. Ribas, "Majn logic synthesis for emerging technology," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 3, Mar 2020.

[28] S. Muroga, *Threshold logic and its applications*, 1st ed. John Wiley & Sons, 1972.

[29] S. Mozaffari, S. Tragoudas, and T. Haniotakis, "A generalized approach to implement efficient CMOS-based threshold logic functions," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 65, no. 3, pp. 946–959, Mar. 2018.

[30] A. Neutzling, M. Martins, V. Callegaro, A. Reis, and R. Ribas, "A Simple and Effective Heuristic Method for Threshold Logic Identification," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, May 2018.

[31] C.-C. Lin, C.-H. Liu, Y.-C. Chen, and C.-Y. Wang, "A new necessary condition for threshold function identification," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 1, pp. 1–1, Jan. 2020.

[32] V. Beiu, J. Quintana, and M. Avedillo, "VLSI implementations of threshold logic - a comprehensive survey," *IEEE Trans. on Neural Networks*, vol. 14, no. 5, Sep. 2003.

[33] L. Gao, F. Alibart, and D. Strukov, "Programmable CMOS/memristor threshold logic," *IEEE Trans. on Nanotechnology*, vol. 12, no. 2, pp. 115–119, Mar. 2013.

[34] N. Nukala, N. Kulkarni, and S. Vrudhula, "Spintronic threshold logic array (stla) - a compact, low leakage, non-volatile gate array architecture," in *Proc. of the Int'l Symp. on Nanoscale Architectures*, 2012.

[35] Z. He and D. Fan, "Energy efficient reconfigurable threshold logic circuit with spintronic devices," *IEEE Trans. on Emerging Topics in Computing*, vol. 5, no. 2, pp. 223–237, Apr.-June 2017.

[36] A. Neutzling, M. Martins, R. Ribas, and A. Reis, "A constructive approach for threshold logic circuit synthesis," in *Proc. of the Int'l Symp. on Circuits and Systems (ISCAS)*, 2014.

[37] M. Martins, R. Ribas, and A. Reis, "Functional composition: A new paradigm for performing logic synthesis," in *Proc. of Int'l Symp. on Quality Electronic Design (ISQED)*, 2012, pp. 236–242.

[38] A. Neutzling, M. Martins, R. Ribas, and A. Reis, "Synthesis of threshold logic gates to nanoelectronics," in *Proc. of ymp. on Integrated Circuits and Systems Design (SBCCI)*, 2013.

[39] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On rewiring and simplification for canonicity in threshold logic circuits," in *Proc. of the Int'l Conf. on Computer-Aided Design*, 2011.

[40] V. Annampedu and M. Wagh, "Decomposition of threshold functions into bounded fan-in threshold functions," *Information and Computation.*, vol. 227, pp. 84–101, June 2013.

[41] N. Kulkarni, J. Yang, J.-S. Seo, and S. Vrudhula, "Reducing power, leakage, and area of standard-cell asics using threshold logic flip-flops," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 9, pp. 2873–2886, Sep. 2016.

[42] A. Neutzling, J. Matos, A. Reis, R. Ribas, and A. Mishchenko, "Threshold logic synthesis based on cut pruning," in *Proc. of the Int'l Conf. on Computer-Aided Design (ICCAD)*, 2015.

[43] Y.-C. Chen, L.-C. Zheng, and F.-L. Wong, "Optimization of threshold logic networks with node merging and wire replacement," *ACM Trans. on Design Automation of Electronic Systems*, vol. 24, no. 6, Oct. 2019.

[44] Y.-C. Chen, H.-J. Chang, and L.-C. Zheng, "Don't-care-based node minimization for threshold logic networks," pp. 1–6, 2020.

[45] C. Albrecht, "IWLS 2005 benchmarks," in *Int'l Workshop on Logic and Synthesis (IWLS)*, 2005.

[46] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for lut-based fpgas," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, Feb. 2007.

[47] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Tech. Dig. of the Int'l Workshop on Logic and Synthesis (IWLS)*, 2015.

[48] J. Pistorius, M. Hutton, A. Mishchenko, and R. Brayton, "Benchmarking method and designs targeting logic synthesis for FPGA," in *Tech. Dig. of the Int'l Workshop on Logic and Synthesis*, 2007.

[49] K. Navi, A. Chabi, and S. Sayedsalehi, "A novel seven input majority gate in quantum-dot cellular automata," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 1, pp. 84–89, Jan. 2012.

[50] A. Roohi, H. Khademolhosseini, S. Sayedsalehi, and K. Navi, "A symmetric quantum-dot cellular automata design for 5-input majority gate," *Journal of Computational Electronics*, vol. 13, no. 3, pp. 701–708, June 2014.

[51] P. Shabadi, A. Khitun, P. Narayanan, M. Bao, I. Koren, K. Wang, and C. Moritz, "Towards logic functions as the device," in *Proc. of the Int'l Symp. on Nanoscale Architectures (NANOARCH)*, 2010, pp. 11–16.

[52] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, May 2016.

[53] M. Soeken, L. Amarú, P. Gaillardon, and G. De Micheli, "Exact synthesis of majority-inverter graphs and its applications," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1842–1855, Nov. 2017.

[54] C. Campos, A. Marciano, O. Neto, and F. Torres, "USE: A universal, scalable, and efficient clocking scheme for qca," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 513–517, Mar. 2016.

[55] S. Akers, "Synthesis of combinational logic using three-input majority gates," in *Proc. of the Symp. on Switching Circuit Theory and Logical Design*, 1962, pp. 149–158.