Majority Logic Synthesis

(Invited Paper)

Luca Amarù[1] Eleonora Testa[2] Miguel Couceiro[3] Odysseas Zografos[4]
Giovanni De Micheli[2] Mathias Soeken[2]
[1]Synopsys Inc., USA [2] EPFL, Switzerland [3]Université de Lorraine, France [4]imec, Belgium

ABSTRACT

The majority function $\langle xyz \rangle$ evaluates to true, if at least two of its Boolean inputs evaluate to true. The majority function has frequently been studied as a central primitive in logic synthesis applications for many decades. Knuth refers to the majority function in the last volume of his seminal *The Art of Computer Programming* as "probably the most important ternary operation in the entire universe." Majority logic sythesis has recently regained signficant interest in the design automation community due to nanoemerging technologies which operate based on the majority function. In addition, majority logic synthesis has successfully been employed in CMOS-based applications such as standard cell or FPGA mapping.

This tutorial gives a broad introduction into the field of majority logic synthesis. It will review fundamental results and describe recent contributions from theory, practice, and applications.

1 INTRODUCTION

The *majority function* [1] of three Boolean variables x, y, and z, denoted $\langle xyz \rangle$, evaluates to true if and only if at least two of the three variables are true. The majority function is self-dual [2] and can be expressed in disjunctive and conjunctive normal form as

$$\langle xyz \rangle = xy \vee xz \vee yz = (x \vee y)(x \vee z)(y \vee z). \tag{1}$$

Setting any variable to 0 gives the conjunction of the other two variables, and analogously one obtains the disjunction by setting any variable to 1, i.e.,

$$\langle 0xy \rangle = x \wedge y \quad \text{and} \quad \langle 1xy \rangle = x \vee y. \tag{2}$$

The majority function can be generalized to n variables

$$\langle x_1 x_2 \dots x_n \rangle = \left[x_1 + x_2 + \dots + x_n > \tfrac{n-1}{2} \right], \tag{3}$$

where n is odd as a special case of the threshold function [3, 4]. The function evaluates to true if at least $\lceil \frac{n}{2} \rceil$ variables are true. We refer to a *majority expression* as a multi-level expression that consists of majority functions with different number of inputs (fan-in), Boolean variables, and constants. If all majority functions in a majority expression have n inputs, we call the expression a *majority-n expression*.

Majority expressions can be represented as a graph, in which nodes represent constants, variables, and majority operations and edges connect operations to their arguments. Edges can be regular or complemented depending on whether the respective argument is complemented or not. We call such graphs *Majority-Inverter Graphs* (MIGs, [5]) following the nomenclature of *And-Inverter Graphs* (AIGs, [6, 7]). Fig. 1 shows the MIG of a full adder with

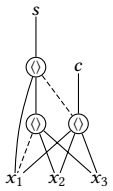

Figure 1: MIG of a full adder with sum s and carry-out c

$$s = \langle x_1 \langle \bar{x}_1 x_2 x_3 \rangle \overline{\langle x_1 x_2 x_3 \rangle} \rangle = x_1 \oplus x_2 \oplus x_3 \text{ and } c = \langle x_1 x_2 x_3 \rangle. \tag{4}$$

Note that the expression of the carry-out c is shared in the expression for the sum s. In the MIG, input nodes are drawn as rectangles, majority operations as circles, regular edges as solid strokes, and complemented edges as dashed strokes.

The explicit consideration of the majority function in Boolean equations has first been suggested by Lindaman [8]. He proposes an algebraic expansion to translate expressions for n-variable Boolean functions of the form

$$x \bigvee_{i=1}^{n-1} f_i \vee \bar{x} \bigwedge_{i=n}^{2(n-1)} f_i \tag{5}$$

into majority-3 expressions. In (5), the f_i are Boolean functions that do not depend on x. It is exemplarily shown in [8] that this expansion can lead to much smaller expressions when compared to directly replacing AND and OR gates using the equations given in (2).

Miiller and Winder [9] presented a synthesis algorithm based on Karnaugh-Veigh diagrams similar to the algorithms used to derive sum-of-product expressions. To synthesize a function f, the algorithm first tries to express f as $\langle f_1 f_2 f_3 \rangle$ by heuristically selecting a majority operation to express f_1 that minimizes $|f \oplus f_1|$, i.e., it agrees closest with the function values of f. This choice implies constraints on function values of f_2 and f_3, on which the algorithm recurs. The algorithm may backtrack if subexpressions result which cannot be realized by a majority function. The algorithm already becomes cumbersome for moderate n but offers a convenient pen-and-paper approach for small functions (see also [10]).

Akers [11] presented an embedding of incompletely specified Boolean functions, represented as truth tables, into monotone self-dual functions [12, 13]. These functions can then be reduced into what he calls a *reduced unitized table*. This is used as a starting

ICCAD'18, San Diego, CA, USA

DOI: https://doi.org/10.1145/3240765.3267501

point to derive various different majority expressions: i) canonical majority expressions, ii) canonical majority-3 expressions, and iii) possibly small (but not guaranteed optimal) majority-3 expressions.

A lot of synthesis algorithms for majority expressions have been presented in the field of quantum-dot cellular automata (QCA, [14, 15]). The basic blocks in QCA are the identity, negation, and majority-3, and therefore QCA circuits inherently realize majority-3 expressions [16]. Initial research [17–19] targeted the realization of 3-variable functions as majority-3 expressions based on the methods from Miiller and Winder [9] and Akers [11]. Several groups [20–22] have then extended this method to synthesize multi-level multi-output Boolean functions. Their techniques start from a minimized algebraically factored circuit [23]. The circuit is then decomposed into nodes that have at most three inputs; if a node represents the majority function it is directly mapped, otherwise the previous mentioned techniques are used to find a good majority-3 expression. The various algorithms differ mainly in how the initial circuit is preprocessed and decomposed and in how nodes are translated into majority expressions. The algorithms have been further extended [24] to consider technology-aware cost metrics [25, 26] or to consider four inputs in the decomposed nodes [27] and have been optimized using genetic algorithms [28, 29].

In this tutorial we review the state-of-the-art in several topics of majority logic synthesis. The next section discusses how practical algorithms can be implemented for optimizing majority-inverter graphs based on majority-3 gates. Section 3 reviews the theory of decomposing majority-n gates into smaller ones and illustrates a practical algorithm based on binary decision diagrams. Section 4 discusses the effect of the majority operation in normal forms to express Boolean functions. Finally, Section 5 concludes with showing applications in nanoemerging technologies using devices that operate based on the majority function.

2 PRACTICAL MAJORITY LOGIC SYNTHESIS

In this part of the tutorial, we present practical data structures and manipulation frameworks for majority logic synthesis. We introduce *Majority-Inverter Graphs* (MIGs), as natural representation form for majority logic, and a dedicated Boolean algebra to perform optimization and reasoning taks with MIGs. We discuss how primitive axioms of the MIG Boolean algebra can be used, alone or combined together, to run efficient majority logic synthesis.

2.1 Majority-Inverter Graphs

MIGs are a logic representation form based on majority and inverter operators [5]. An MIG is a logic network consisting of 3-input majority nodes and regular/complemented edges. MIGs can emulate traditional *And Or Inverter Graphs* (AOIGs) by fixing to 0 (AND) or to 1 (OR) one input of the majority nodes. Fig. 2 depicts two logic representation examples for MIGs. They are obtained by translating their *And Or Inverter Graphs* (AOIGs) representations into MIGs, using the aforementioned strategy.

2.2 MIG Boolean Algebra

To natively operate on MIGs, a set of bidirectional transformations, named Ω, is introduced in [5] and reported hereafter.

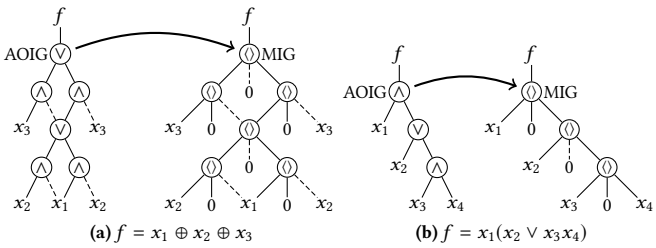

Figure 2: Examples of MIG representations (right) for (a) $f = x \oplus y \oplus z$ and (b) $g = x(y \vee uv)$ derived by transposing their AOIG representations (left). Complement attributes are represented by dashed edges.

$$
\begin{array}{ll}
\Omega.\mathrm{C} : \text{Commutativity} & \langle xyz\rangle = \langle yxz\rangle = \langle zyx\rangle \\
\Omega.\mathrm{M} : \text{Majority} & \langle xxz\rangle = x \qquad \langle x\bar{x}z\rangle = z \\
\Omega.\mathrm{A} : \text{Associativity} & \langle xu\langle yuz\rangle\rangle = \langle zu\langle yux\rangle\rangle \\
\Omega.\mathrm{D} : \text{Distributivity} & \langle xy\langle uvz\rangle\rangle = \langle\langle xyu\rangle\langle xyv\rangle z\rangle \\
\Omega.\mathrm{I} : \text{Inverter propagation} & \langle \bar{x}\bar{y}\bar{z}\rangle = \overline{\langle xyz\rangle}
\end{array}
\tag{6}
$$

By using Ω, it is possible to optimize an MIG with respect to a desired metric. For example, majority Ω.M enables size and depth reduction when applied from left to right. Also, distributivity Ω.D enables depth reduction when applied from left to right and z is a critical variable. On the other hand, distributivity Ω.D applied from right to left enables size reduction. In a similar fashion, the remaining transformations in Ω are also useful in MIG optimization.

2.3 MIG Boolean Algebra Derived Theorems

Several other complex rules are derivable from Ω. Among the ones we encountered, three rules derived from Ω are of particular interest to logic optimization. We refer to them as Ψ and are described hereafter. In the following, the symbol $f_{x/y}$ represents a replacement operation, i.e., it replaces x with y in all its appearence in f.

$$
\begin{array}{ll}
\Psi.\mathrm{R} : \text{Relevence} & \langle xyz\rangle = \langle xyz\rangle_{x/\bar{y}} \\
\Psi.\mathrm{C} : \text{Compl. assoc.} & \langle xy\langle y\bar{u}z\rangle\rangle = \langle xu\langle yxz\rangle\rangle \\
\Psi.\mathrm{S} : \text{Substitution} & \langle xyz\rangle = \langle v\langle\bar{v}\langle xyz\rangle_{v/u}u\rangle\langle\bar{v}\langle xyz\rangle_{v/\bar{u}}\bar{u}\rangle\rangle,
\end{array}
\tag{7}
$$

The first rule, relevance Ψ.R, replaces reconvergent variables with their neighbors. For example, consider the function $f = \langle xy\langle w\bar{z}\langle xyz\rangle\rangle\rangle$. Variables x and y are reconvergent because they appear in both the bottom and the top majority operators. In this case, relevance Ψ.R replaces x with $\bar{y}$ in the bottom majority as $f = \langle xy\langle w\bar{z}\langle\bar{y}yz\rangle\rangle\rangle$. This representation can be further reduced to $f = \langle xyw\rangle$ by using Ω.M.

The second rule, complementary associativity Ψ.C, deals with variables appearing in both polarities. Its rule of replacement is $\langle xu\langle y\bar{u}z\rangle\rangle = \langle xu\langle yxz\rangle\rangle$ as depicted by (7).

The third rule, substitution Ψ.S, extends variable replacement to the non-reconvergent case. We refer the reader to Fig. 3 for an example about substitution Ψ.S applied to a 3-input parity function.

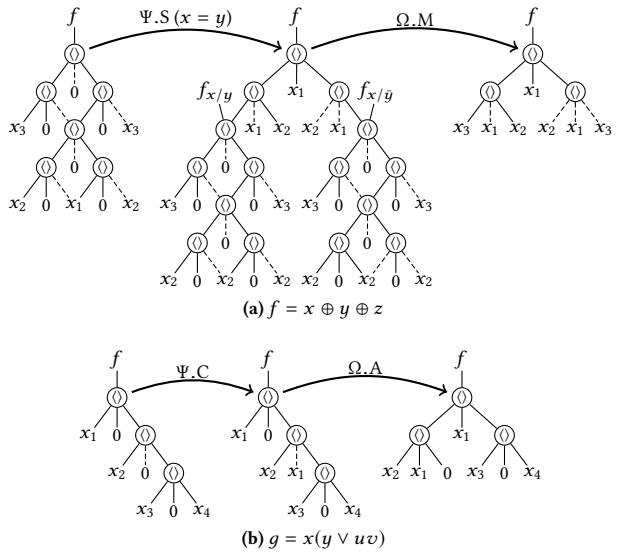

Figure 3: Examples of optimized MIG representations (right).

2.4 MIG Optimization Example

In an automated MIG optimization tool, Ω and Ψ transformations are iterated in sequence, with a sense (left-to-right or right-to-left) acting in accordance to the chosen target metric. We refer the reader to [5] for more details on the use of Ω and Ψ in MIG optimization.

Fig. 3 shows how examples from Fig. 2 can be optimized using Ω and Ψ rules.

3 DECOMPOSING MAJORITY FUNCTIONS

In this part of the tutorial, we explore the functional decomposition of majority-n networks, which permits the use of majority primitives with n inputs (where n is odd). We particularly focus on the decomposition into majority-3 as this is the main building block of several emerging nanotechnologies. After reviewing the decomposition methods from the classic literature, we discuss a more recent techniques that exploit binary decision diagrams. We conclude with the challenge of finding minimum-size decompositions.

3.1 State-of-the-art

The problem of expressing majority-n using majority-of-three has already been studied in the 1960s. In [30], Amarel et al. investigated "*How best can the 5-argument majority function be realized with a network of 3-input majority gates?*" The focus was on finding the minimum number of 3-input majorities to build majority-5, but larger n were also considered [30]. In the remainder, we will refer to the minimum number of majority-of-three to build a majority-n as $M(n)$. It is surprising that, as of today, $M(n)$ is known only for 5- and 7-input majorities, while the minimum realization of majority-9 (and larger n) in terms of majority-3 is still under investigation.

Other works have focused on $M(n)$, but they have only considered its asymptotic bounds [31]. A quasi-linear construction follows from sorter networks [32]. We simply sort the n bits and pick the one that ends up in the middle position. Sorter networks consist of comparators, and each comparator can be composed with 2 majority-3 operations. Let $S(n)$ be the optimum number of comparators in a sorter network that sorts n elements. Then an upper bound on the number of majority-3 operations is $u_S(n) \leq 2S(n)$.

Dor and Zwick showed that less than $2.942n$ comparisons are necessary to select the median value from a set of n numbers, without the need to sort them [33]. Applying it directly, it would lead to an upper bound of $5.884n$ majority-3 operations to decompose majority-n. However, this number needs to be treated more carefully, since their analysis is based on the *comparison model* in which *only* the number of comparators are counted and all other operations are considered free.

When applying the discussed constructions to small values for n, the resulting majority graphs are still very large. For example, the majority-7 function can be constructed using 42 majority-3 operations according to median selection construction, while it is known that $M(7) = 7$. Even if sorter networks provide an alternative construction that provides a quasi-linear bound [34], for small n the construction can yield better results compared to median selection. To follow up with the previous example, the majority-7 function can be constructed using 32 majority-3 operations based on the sorter network construction.

In this tutorial, we discuss an alternative construction based on *Binary Decision Diagrams* (BDDs, [35]) as first presented in [36]. We show that for monotone Boolean functions the Shannon decomposition, which is used in the construction of BDDs, can be expressed using the majority function.

3.2 Transforming BDDs into Majority Graphs

Binary Decision Diagrams (BDDs, [35]) are connected directed acyclic graphs, in which each node represents a Boolean function. Two terminal nodes labeled '$\bot$' and '$\top$' represent the constant functions 0 and 1, and all nonterminal nodes are labeled 'i' for some $1 \leq i \leq n$ and connect their two successor nodes f_{x_i} and $f_{\bar{x}_i}$ using Shannon's decomposition:

$$f = x_i \, ? \, f_{x_i} : f_{\bar{x}_i} = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i} \tag{8}$$

Here, the cofactors f_{x_i} and $f_{\bar{x}_i}$ are the functions that are obtained by replacing x_i with 1 and 0 in f, respectively. Each BDD has one node without parents representing the function, called the root. One can express any Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ in terms of its cofactors using (8). If f is monotone, i.e., $f_{\bar{x}_i} \bar{f}_{x_i} = 0$ for all i, it is possible to use the majority function for decomposition [12]:

$$f = \langle x_i f_{x_i} f_{\bar{x}_i} \rangle = x_i f_{x_i} \oplus x_i f_{\bar{x}_i} \oplus f_{x_i} f_{\bar{x}_i} \tag{9}$$

The two cofactors f_{x_i} and $f_{\bar{x}_i}$ commute in (9), but they do not in (8). Plus, the cofactor operation preserves monotonicity in (8). Thus, (9) can iteratively be applied to the whole BDD.

The main idea presentd in [36] is that, due to these properties,

$$f = x_i \, ? \, f_{x_i} : f_{\bar{x}_i} = \langle x_i f_{x_i} f_{\bar{x}_i} \rangle \tag{10}$$

one can replace each "Shannon node" by a "majority node" in the BDD of monotone functions. The replacement allows us to generate majority graphs for monotone functions $f(x_1, \ldots, x_n)$ directly from their BDDs using the discussed transformation rule.

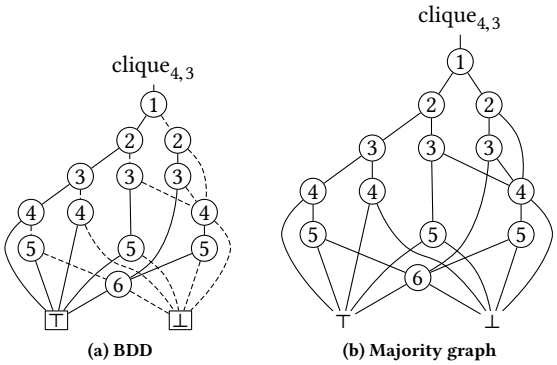

Figure 4: Diagrammatic notation of a BDD (a) and majority graph (b) for the function $\text{clique}_{4,3}$

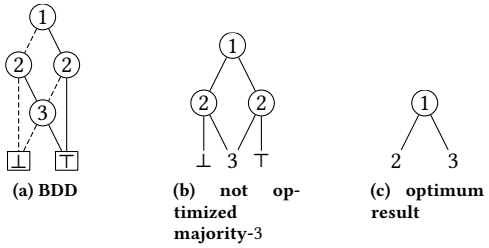

Figure 5: Majority-3 from its optimal BDD

Example 3.1. We show how to apply the transformation to the BDD in Fig. 4. By just translating every BDD node into a MAJ node as in (10), one obtains the majority graph depicted in Fig. 4(b) directly from its BDD (Fig. 4(a)).

3.3 Minimum-size Decomposition and Upper Bounds

The proposed approach can be used to map the majority-n function into MIGs with majority-3 operations. For example, for the majority-3 function $\langle x_1 x_2 x_3 \rangle$, the BDD and its corresponding majority graph are shown in Fig. 5(a) and (b), respectively. Note that the majority expression that Fig. 5(b) represents is $\langle x_1 \langle x_2 0 x_3 \rangle \langle x_2 1 x_3 \rangle \rangle$, which is of course far from optimal. The distributivity rule applies to this expression, giving $\langle \langle x_1 01 \rangle x_2 x_3 \rangle = \langle x_1 x_2 x_3 \rangle$; its diagrammatic notation is shown in Fig 5(c). This example was rather trivial. The same approach is applied in [36] to rewrite majority-5 and majority-7 into their optimum MIGs using the identities presented in [36, 37]. The majority-5 optimization is described in Fig. 6; all the details can be found in [36].

The proposed synthesis method from a BDD suggests an upper bound $u_{\mathrm{B}}(n)$ for the majority-n function. The majority-n function $\langle x_1 \dots x_n \rangle$ can be realized using a majority graph with at most $u_{\mathrm{B}}(n) \leq \left(\left\lceil \frac{n}{2} \right\rceil\right)^2 - 1$ majority operations. Both the construction on sorter networks and median selection have asymptotically better upper bounds compared to the quadradic bound from the BDD construction. However, when actually calculating the numbers for small instances with $n \leq 17$, the BDD approach yields the smallest values (see Table 1, which also shows the known optimum results up to $n = 7$ that were confirmed using exhaustive enumeration [38]). The approach is therefore a good starting point for finding compact majority-n realizations for small n. The results obtained using BDD + optimization rules are listed in Table 1 as "Optimized BDD". The method in [36] is able to obtain (i) the optimum known results for majority-5 and majority-7, and (ii) the best result for majority-9. One may be able to derive a general derivation procedure to obtain optimum or close to optimum majority-n realizations for $n \geq 9$.

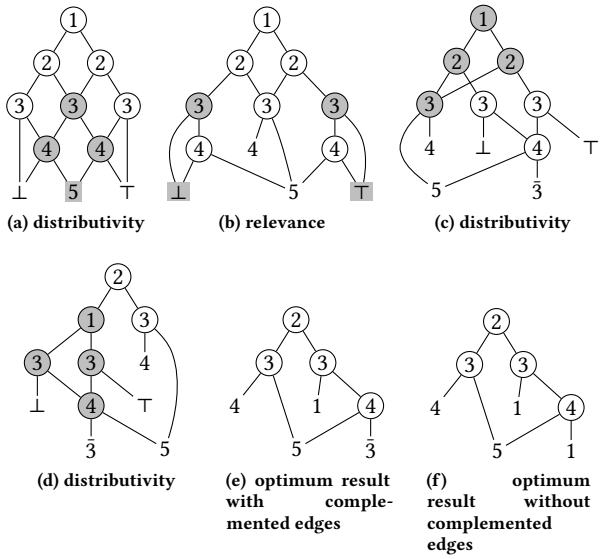

Figure 6: Decomposing majority-5 into majority-3

Table 1: Upper bounds on the number of majority-3 operations to realize majority-n

n	3	5	7	9	11	13	15	17
Optimum ($M(n)$)	1	4	7					
Optimized BDDs	1	4	7	**15**				
BDDs	3	8	15	24	35	48	63	80
Sorter networks	6	18	32	50	70	90	112	142
Median selection*	18	30	42	53	65	77	89	101

* These numbers are based on the number of comparators in the construction of [33], but do not take other operations into account.

4 EFFICIENT NORMAL FORM SYSTEMS

We focus on the efficient representation of Boolean functions by certain systems of stratified terms and called Normal Form Systems (NFSs). These induce sets of terms that are characterized by the order in which connectives occur. Well-known examples include the Disjunctive Normal Form (DNF), Conjunctive Normal Form (CNF), or median NFS (MNF) that has as non-trivial connective the median, that is the ternary generator of the clone SM of self-dual monotone Boolean functions. It was proven in [39] that the

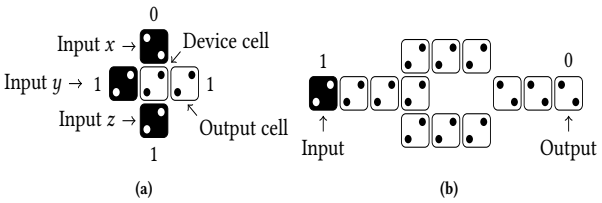

Figure 7: (a) QCA layout for majority, (b) inverter

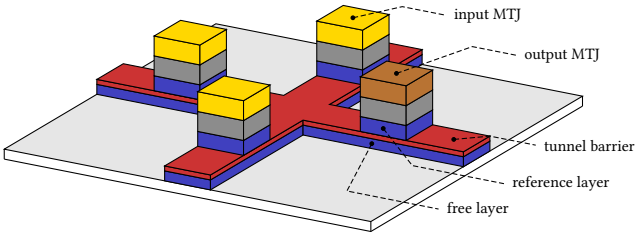

Figure 8: Schematic of STMG [43]. The three MTJ inputs are visible in yellow, while the output is the orange block.

median normal form system is polynomially more efficient than the DNF, CNF, polynomial and dual polynomial normal form systems, associated with the generators of the clones Λ of all conjunctions, the clone V of all disjunctions, and the clone L of all linear functions ($\wedge$, $\vee$, and $\oplus$, respectively). However, these results were obtained for certain generators of these clones. A natural question to ask is whether these results still hold for arbitrary generators. It is known that clones are finitely generated; however, and as it is the case for the clone SM of monotone self-dual functions, several generators (or sets of generators) may exist. For instance, SM is generated by the 5-ary median (and, in fact, by any $(2n+1)$-ary median). Are the NFSs associated with the ternary majority and the 5-ary majority equivalently efficient? In fact, it can be shown that they are. Remark, for instance, that

$$\langle xyz\rangle = \langle xyz01\rangle,$$
$$\langle xyztu\rangle = \langle\langle xyz\rangle t\langle\langle xyu\rangle uz\rangle\rangle.$$

Both equalities allow the efficient conversion from terms involving majority-5 into terms involving majority-3, and reciprocally.

Furthermore, in general, it can be shown that the choice of generator for SM does not impact the efficiency of the NFS associated with it. These results motivate the study of other efficient NFS, that is, NFS that are equivalent to the median normal form systems. These include, for instance, the Sheffer NF that is generated by the Sheffer function $x \uparrow y \equiv \neg(x \wedge y)$, one of the two generators of minimal arity of the clone Ω of all Boolean functions, the other being Peirce's arrow, $x \downarrow y \equiv \neg(x \vee y)$ (see, e.g., [40]).

In [41] such efficient NFSs were studied and were proven to be equivalent for generators of minimal arity. A natural question to ask, just as for the MNF, is whether these results still hold for arbitrary generators. In fact, it can be shown that any generator of Ω yields an efficient NFS, and similarly for other clones associated with non-associative connectives. Furthermore, the choice of generator for the clones associated with non-efficient NFSs, namely, the clones of disjunctions, conjunctions, and linear functions, does not impact the efficiency results.

5 EMERGING NON-CHARGE-BASED MAJORITY LOGIC TECHNOLOGIES

In this section we describe QCA and STMG. Both technologies use the 3-input majority gate as their main building block.

QCA technology is based on the interaction of QCA cells: each cell consists of four quantum dots and two free electrons. The free electrons can tunnel between the dots, which are coupled by tunnel barriers. Coulomb repulsion forces the electrons in opposite corners of the cell, thus producing two energetically equivalent polarizations, i.e., $P = 1$ and $P = -1$. The two polarizations represent the logic values 1 and 0, respectively. QCA technology is functionally complete, and the fundamental logic element of QCA is the three-inputs majority gate [42]. Figs. 7a and 7b show the layout of a QCA majority gate and inverter, respectively. For the majority gate the polarization of the central logic cell, called device cell, is the majority of the three inputs; the output cell follows the polarization of the device cell. In the inverter case, the input wire is first branched in two offset wires; both will have the same polarization as the input due to aligning effects. Anti-aligning effects at the second joint control the polarization of the next cell, causing an inversion of the input signal.

STMG is a three-input majority gate that can ensure small area, low power, non-volatility, reconfigurability, and radiation hardness [43]. STMG is driven by *spin transfer torque* (STT, [44]); its schematic is shown in Fig. 8. Four *magnetic tunnel junctions* (MTJ) share one cross-shaped free layer. Three MTJs (yellow) write the input states via STT, while the forth MTJ (orange) reads the output state via tunnel magnetoresistance. The magnetic domains are mainly driven by domain wall automotion [45], and the magnetization orientation (up/down) in the free layer carries the bit information (0 or 1). Despite of potential advantages as small area and low power, there is a lack of an efficient *spin torque inverter* (STI) concept which would be necessary to implement circuits. An STI implementation has been proposed in [46], but it cannot be realized with the state-of-the-art technology for magnetic materials. In [47], the lack of STI has been overcome by demonstrating inverter-free circuits, obtained using MIGs as intermediate data structure.

Acknowledgments. The authors like to thank Pierre-Emmanuel Gaillardon, Winston Haaswijk, Alan Mishchenko, and Heinz Riener, who contributed to various works reviewed in this tutorial. This research was supported by the Swiss National Science Foundation (200021-169084 MAJesty) and by the European Research Council in the project H2020-ERC-2014-ADG 669354 CyberCare.

REFERENCES

[1] R. L. Wigington, "A new concept in computing," *Proceedings of the IRE*, vol. 47, no. 4, pp. 516–523, 1959.
[2] D. E. Knuth, *The Art of Computer Programming, Volume 4A.* Addison-Wesley, 2011.
[3] C. C. Elgot, "Truth functions realizable by single threshold organs," in *Foundations of Computer Science*, 1960, pp. 225–245.
[4] R. O. Winder, "More about threshold logic," in *SWCT*, 1961, pp. 55–64.

[5] L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Design Automation Conference*, 2014, pp. 194:1–194:6.

[6] L. Hellerman, "A catalog of three-variable Or-invert and And-invert logical circuits," *IEEE Trans. Electronic Computers*, vol. 12, no. 3, pp. 198–223, 1963.

[7] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, 2002.

[8] R. Lindaman, "A theorem for deriving majority-logic networks within an augmented Boolean algebra," *IRE Trans. Electronic Computers*, vol. 9, no. 3, pp. 338–342, 1960.

[9] H. S. Miller and R. O. Winder, "Majority-logic synthesis by geometric methods," *IRE Trans. Electronic Computers*, vol. 11, no. 1, pp. 89–90, 1962.

[10] O. A. Horna, "A geometric synthesis method of three-input majority logic networks," *IEEE Trans. Electronic Computers*, vol. 14, pp. 475–481, 1965.

[11] S. B. Akers Jr., "Synthesis of combinational logic using three-input majority gates," in *Symp. on Switching Circuit Theory and Logical Design*, 1962, pp. 149–157.

[12] ——, "A truth table method for the synthesis of combinational logic," *IEEE Trans. Electronic Computers*, vol. 10, no. 4, pp. 604–615, 1961.

[13] R. C. De Vries, "Minimal sets of distinct literals for a logically passive function," *Journal of the ACM*, vol. 3, pp. 431–443, 1971.

[14] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *Journal of Applied Physics*, vol. 75, pp. 1818–1825, 1994.

[15] I. Amlani, A. O. Orlov, G. Toth, G. H. Bernstein, C. S. Lent, and G. L. Snider, "Digital logic gate using quantum-dot cellular automata," *Science*, vol. 284, pp. 289–291, 1999.

[16] G. L. Snider, A. O. Orlov, I. Amlani, G. H. Bernstein, C. S. Lent, J. L. Merz, and W. Porod, "Quantum-dot cellular automata: line and majority logic gate," *Japanese Journal of Applied Physics*, vol. 38, pp. 7227–7229, 1999.

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

[18] K. Walus, G. Schulhof, G. A. Jullien, R. Zhang, and W. Wang, "Circuit design based on majority gates for applications with quantum-dot cellular automata," in *Asilomar Conf. on Signals, Systems and Computers*, 2004, pp. 1354–1357.

[19] S. Rai, "Majority gate based design for combinational quantum cellular automata (QCA) circuits," *SSST*, vol. 40, pp. 222–224, 2008.

[20] R. Zhang, P. Gupta, and N. K. Jah, "Synthesis of majority and minority networks and its application to QCA, TPL and SET based nanotechnologies," in *VLSI Design*, 2005, pp. 229–234.

[21] Z. Huo, Q. Zhang, S. Haruehanroengra, and W. Wang, "Logic optimization for majority gate-based nanoelectronic circuits," in *Int'l Symp. on Circuits and Systems*, 2006.

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

[23] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[24] S. Vemry, P. Wang, and M. Niamat, "Majority logic gate synthesis approaches for post-cmos logic circuits: A review," in *Int'l Conf. on Electro/Information Technology*, 2014, pp. 284–289.

[25] 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, 2010.

[26] M. G. A. Martins, V. Ceelgaro, F. S. Marranghello, R. P. Ribas, and A. I. Reis, "Majority-based logic synthesis for nanometric technologies," in *Int'l Conf. on Nanotechnology*, 2014, pp. 256–261.

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

[28] M. R. Bonyadi, S. M. R. Azghadi, N. M. Rad, K. Navi, and E. Afjei, "Logic optimization for majority gate-based nanoelectronic circuits based on genetic algorithm," in *ICEE*, 2007, pp. 1–5.

[29] M. A. Tehrani, K. Navi, and A. Kia-kojoori, "Multi-output majority gate-based design optimization by using evolutionary algorithm," in *Swarm and Evolutionary Computation*, 2013, pp. 25–30.

[30] S. Amarel, G. E. Cooke, and R. O. Winder, "Majority gate networks," *IEEE Trans. Electronic Computers*, vol. 13, no. 1, pp. 4–13, 1964.

[31] A. S. Kulikov and V. V. Podolskii, "Computing majority by constant depth majority circuits with low fan-in gates," in *Symposium on Theoretical Aspects of Computer Science*, 2017, pp. 49:1–49:14.

[32] D. E. Knuth, *The Art of Computer Programming, Volume 3, Second Edition*. Addison-Wesley, 1998.

[33] D. Dor and U. Zwick, "Selecting the median," *SIAM Journal on Computing*, vol. 28, no. 5, pp. 1722–1758, 1999.

[34] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp, "Twenty-five comparators is optimal when sorting nine inputs (and twenty-nine for ten)," in *Int'l Conf. on Tools with Artificial Intelligence*, 2014, pp. 186–193.

[35] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[36] E. Testa, M. Soeken, L. Amarù, W. Haaswijk, and G. D. Micheli, "Mapping monotone Boolean functions into majority," to be published.

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

[38] M. Soeken, L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Exact synthesis of majority-inverter graphs and its applications," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1842–1855, 2017.

[39] M. Couceiro, S. Foldes, and E. Lehtonen, "Composition of Post classes and normal forms of Boolean functions," *Discrete Mathematics*, vol. 306, no. 24, pp. 3223–3243, 2006.

[40] E. Żyliński, "Some remarks concerning the theory of deduction," *Fundamenta Mathematicae*, vol. 7, no. 1, pp. 203–209, 1925.

[41] M. Couceiro, P. Mercuriali, and R. Péchoux, "Comparing the efficiency of normal form systems to represent Boolean functions," urlhttps://hal.inria.fr/hal-01551761.

[42] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *Journal of Applied Physics*, vol. 75, no. 3, pp. 1818–1825, 1993.

[43] D. E. Nikonov, G. I. Bourianoff, and T. Ghani, "Proposal of a spin torque majority gate logic," *IEEE Electron Device Letters*, vol. 32, no. 8, pp. 1128–1130, 2011.

[44] L. Berger, "Emission of spin waves by a magnetic multilayer traversed by a current," *Physical Review B*, vol. 54, no. 13, pp. 9353–9358, 1996.

[45] D. E. Nikonov, S. Manipatruni, and I. A. Young, "Automotion of domain walls for spintronic interconnects," *Journal of Applied Physics*, vol. 115, no. 21, p. 213902, 2014.

[46] D. E. Nikonov, G. I. Bourianoff, and T. Ghani, "Nanomagnetic circuits with spin torque majority gates," in *Int'l Conf. on Nanotechnology*, 2011, pp. 1384–1388.

[47] E. Testa, O. Zografos, M. Soeken, A. Vaysset, M. Manfrini, R. Lauwereins, and G. De Micheli, "Inverter propagation and fan-out constraints for beyond-CMOS majority-based technologies," in *Annual Symp. on VLSI*, 2017, pp. 164–169.