Majority logic synthesis
Luca Amarù, Eleonora Testa, Miguel Couceiro, Odysseas Zografos, Giovanni de Micheli, Mathias Soeken

To cite this version:
Luca Amarù, Eleonora Testa, Miguel Couceiro, Odysseas Zografos, Giovanni de Micheli, et al.. Majority logic synthesis. ICCAD 2018 - IEEE/ACM International Conference on Computer-Aided Design, Nov 2018, San Diego, United States. 10.1145/3240765.3267501 . hal-01925946

HAL Id: hal-01925946
https://inria.hal.science/hal-01925946
Submitted on 2 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Majority logic synthesis

(Embedded tutorial)

Luca Amaru\textsuperscript{1} Eleonora Testa\textsuperscript{2} Miguel Couceiro\textsuperscript{3} Odysseas Zografos\textsuperscript{4} Giovanni De Micheli\textsuperscript{2} Mathias Soeken\textsuperscript{2} \\
\textsuperscript{1}Synopsys Inc., USA \hspace{0.2cm} \textsuperscript{2}EPFL, Switzerland \hspace{0.2cm} \textsuperscript{3}Université de Lorraine, France \hspace{0.2cm} \textsuperscript{4}imec, Belgium

Abstract—The majority function \( (xyz) \) 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 \textit{The Art of Computer Programming} as “probably the most important ternary operation in the entire universe.” Majority logic synthesis has recently regained significant interest in the design automation community due to nanoelectronic 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.

I. INTRODUCTION

The \textit{majority function} \cite{1} of three Boolean variables \( x, y, \) and \( z, \) denoted \( (xyz), \) evaluates to true if and only if at least two of the three variables are true. The majority function is self-dual \cite{2} and can be expressed in disjunctive and conjunctive normal form as

\[
(0xyz) = x \land y \quad \text{and} \quad (1xyz) = x \lor y. \tag{2}
\]

The majority function can be generalized to \( n \) variables

\[
(x_1x_2 \ldots x_n) = \left[x_1 + x_2 + \cdots + x_n > \frac{n-1}{2}\right], \tag{3}
\]

where \( n \) is odd as a special case of the threshold function \cite{3}, \cite{4}. The function evaluates to true if at least \( \left[ \frac{n}{2} \right] \) variables are true. We refer to \textit{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 \textit{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 \textit{Majority-Inverter Graphs} (MIGs, \cite{5}) following the nomenclature of \textit{And-Inverter Graphs} (AIGs, \cite{6}, \cite{7}). Fig. 1 shows the MIG of a full adder with

\[
s = \langle a \langle b \langle c_{\text{in}} \rangle \rangle \rangle = a \oplus b \oplus c_{\text{in}} \quad \text{and} \quad c_{\text{out}} = \langle c_{\text{in}} \rangle. \tag{4}
\]

Note that the expression of the carry-out \( c_{\text{out}} \) 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.

Fig. 1. MIG of a full adder

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

\[
\bigvee_{i=1}^{2^{n-1}} f_i \odot \bigwedge_{i=n}^{2(n-1)} f_i \tag{5}
\]

into majority-3 expressions. In (5), the \( f_i \) are Boolean functions that do not depend on \( x. \) It is exemplarily shown in \cite{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 \cite{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 \( \langle f_1f_2f_3 \rangle \) 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 \cite{10}).

Akers \cite{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 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], [18], [19] targeted the realization of 3-variable functions as majority-3 expressions based on the methods from Müller and Winder [9] and Akers [11]. Several groups [20], [21], [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 III reviews the theory of decomposing majority-n gates into smaller ones and illustrates a practical algorithm based on binary decision diagrams. Section IV discusses the effect of the majority operation in normal forms to express Boolean functions. Finally, Section V concludes with showing applications in nanoemerging technologies using devices that operate based on the majority function.

II. PRACTICAL MAJORITY LOGIC SYNTHESIS

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.

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

![Fig. 2. Examples of MIG representations (right) for (a) \( f = x \oplus y \oplus z \) and (b) \( g = x(y \lor uv) \) derived by transposing their AOIG representations (left). Complement attributes are represented by bubbles on the edges.](image)
right-to-left) acting in accordance to the chosen target metric. We refer the reader to [5] for more details on the use of \( \Omega \)
(right-to-left) acting in accordance to the chosen target metric. We refer the reader to [5] for more details on the use of \( \Omega \) and \( \Psi \) in MIG optimization.

Fig. 3 shows how examples from Fig. 2 can be optimized using \( \Omega \) and \( \Psi \) rules.

### III. 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.

#### A. 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 based on the sorter network 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.

#### B. 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 \( f_{\bar{x}_i} \) and \( f_{x_i} \) using Shannon’s decomposition:

\[
 f = x_i \oplus f_{x_i} = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i} \quad (8)
\]

Here, the cofactors \( f_{\bar{x}_i} \) and \( f_{x_i} \) 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 \( f : \mathbb{B}^n \rightarrow \mathbb{B} \) in terms of its cofactors using \( (8) \). If \( f \) is monotone, i.e., \( f_{\bar{x}_i}, f_{x_i} = 0 \) for all \( i \), it is possible to use the majority function for decomposition [12]:

\[
 f = \langle x_i f_{x_i}, f_{\bar{x}_i} \rangle = x_i f_{x_i} \oplus x_i f_{\bar{x}_i} \oplus f_{x_i} f_{\bar{x}_i} \quad (9)
\]

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

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

\[
 f = x_i \oplus f_{x_i} = \langle x_i f_{x_i}, f_{\bar{x}_i} \rangle \quad (10)
\]

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.

**Example 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)).

**C. 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 \(x_1x_2x_3\), 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 \(x_1x_2(0x_3\overline{x_3})\), which is of course far from optimal. The distributivity rule applies to this expression, giving \(\langle x_101 \rangle x_2x_3 = \langle x_1x_2x_3 \rangle\); 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 [37], [36]. 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 \(u_B(n)\) for the majority-\(n\) function. The majority-\(n\) function \(x_1 \ldots x_n\) can be realized using a majority graph with at most \(u_B(n) \leq \left(\left\lceil \frac{n}{2} \right\rceil \right)^2 - 1\) majority operations. Both the construction on sorter networks and median selection have asymptotically better upper bounds compared to the quadratic 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 I, 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 I 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 \(n \geq 9\).

**IV. EFFICIENT NORMAL FORM SYSTEMS**

We focus on the efficient representation of Boolean functions. We consider Normal Form Systems (NFSs) as our representation of choice. These induce sets of terms that have a certain form: i.e., certain connectives occur before others. See, e.g., 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 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 $A$ 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

$$
\langle xyz \rangle = \langle xyz01 \rangle,
\langle xyztu \rangle = \langle (xyz)t((xy)zu) \rangle.
$$

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 function $\equiv\neg$ 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 $\Omega$ 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 $\Omega$ 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.

V. 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.