Exact Synthesis of Majority-Inverter Graphs and Its Applications

Mathias Soeken, *Member, IEEE*, Luca Gaetano Amarù, *Member, IEEE*,
Pierre-Emmanuel Gaillardon, *Senior Member, IEEE*,
and Giovanni De Micheli, *Fellow, IEEE*

***Abstract*—We propose effective algorithms for exact synthesis of Boolean logic networks using satisfiability modulo theories (SMTs) solvers. Since exact synthesis is a difficult problem, it can only be applied efficiently to very small functions, having up to six variables. Key in our approach is to use majority-inverter graphs (MIGs) as underlying logic representation as they are simple (homogeneous logic representation) and expressive (contain AND/OR-inverter graphs) at the same time. This has a positive impact on the problem formulation: it simplifies the encoding as SMT constraints and also allows for various techniques to break symmetries in the search space due to the regular data structure. Our algorithm optimizes with respect to the MIG's size or depth and uses different ways to encode the problem and several methods to improve solving time, with symmetry breaking techniques being the most effective ones. We discuss several applications of exact synthesis and motivate them by experiments on a set of large arithmetic benchmarks. Using the proposed techniques, we are able to improve both area and delay after lookup table (LUT)-based technology mapping beyond the current results achieved by state-of-the-art logic synthesis algorithms.**

***Index Terms*—Boolean satisfiability, exact synthesis, logic rewriting, logic synthesis, majority logic.**

I. Introduction

MINIMUM logic networks are of high interest in logic synthesis as they offer strong opportunities for logic optimization. Exact synthesis is the task of finding an optimum logic network representation for a given input specification. Optimality typically addresses the size of the network, i.e., the number of gates, or the depth of the circuit, i.e., the length of the critical path. Exact synthesis is a special instance of the minimum circuit size problem (MCSP, [1]) which is also intensively studied in the computational complexity community and conjectured to be a difficult problem (see [2], [3]).

Manuscript received August 3, 2016; revised November 9, 2016; accepted January 3, 2017. Date of publication February 6, 2017; date of current version October 18, 2017. This work was supported in part by European Research Council under Grant H2020-ERC-2014-ADG 669354 CyberCare, and in part by the Swiss National Science Foundation under Grant 200021-169084 MAJesty. This paper was recommended by Associate Editor S. K. Lim.

M. Soeken and G. De Micheli are with the École Polytechnique Fédéderale de Lausanne, 1015 Lausanne, Switzerland.

L. G. Amarù is with Synopsys, Mountain View, CA 94085 USA.

P.-E. Gaillardon is with the University of Utah, Salt Lake City, UT 84112 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2664059

Implicit enumerative algorithms are considered the most practical ones to solve exact synthesis [4]. These algorithms solve exact synthesis using constraint satisfaction and optimization techniques such as integer linear programming [5] or Boolean satisfiability [6].

The efficiency of implicit enumerative algorithms for exact synthesis depends mainly on: 1) the size or depth of the optimum representation; 2) the encoding of the problem formulation as constraint satisfaction problem; and 3) the ability to trim the search space, e.g., due to symmetric solutions. These three are hard to fulfill equally well at the same time. For example, a rich gate library allows for compact logic networks; however, it impedes to encode the problem and to formulate constraints for symmetry breaking. On the other hand, a minimalist gate library such as NAND gates allows for a simple encoding and a lot of regularity for symmetry breaking constraints, but has larger optimum network representations.

In this paper, we propose to use majority-inverter graphs (MIGs, [7]) as network representation for exact synthesis, as it addresses the three requirements equally well. MIGs are homogeneous logic networks that only have the majority-of-three operation and inversion as logic operations. This makes MIGs simple but yet expressive at the same time, since they contain AND/OR-inverter graphs (AOIGs, see [8], [9]). This allows for a simple encoding while also offering a compact network representation. One striking example is that all functions of three variables are representable with MIGs with at most two logic levels, but require up to four logic levels in AOIGs. In addition, the rich algebraic properties of MIGs support symmetry breaking. Several heuristics for MIG optimization have been proposed in the past (see [7], [10]–[15]).

We solve the task of exact synthesis using MIGs by encoding the problem as an instance of the satisfiability modulo theories (SMT, [16]) problem. We provide an in-depth description of how to encode exact synthesis to SMT for both explicit and implicit representations of the input specification. The exact synthesis algorithm can generate optimum MIGs based on a truth table representation but also based on a nonoptimal MIG. Due to the encoding to SMT formulas the algorithm is general and can in principle be used for many different cost criteria (see also [5]). We show how it can be used to find size-optimum and depth-optimum MIGs. Various symmetry breaking techniques are able to reduce the number of solutions and have a tremendous impact on the runtime.

As the (apparent) complexity of the MCSP limits an efficient applicability of the proposed algorithm to only small functions (with up to six variables), it first becomes truly powerful when embedded into heuristic algorithms. In fact, we show how the proposed exact algorithm can be transformed

into a heuristic itself. We illustrate several logic synthesis applications that make use of exact synthesis as part of their algorithmic description.

Experimental results show that with algorithms based on exact synthesis heavily optimized MIGs can be further optimized in size and depth. We were able to improve the best results of the arithmetic EPFL benchmarks in both cost criteria. The current best results are produced by the strongest and-inverter graph (AIG) and MIG optimization scripts from Berkeley and EPFL groups, hence, our improvements advance the state-of-the-art in logic optimization.

II. Related Work

Synthesis algorithms to find optimal and optimum network realizations can be divided into three categories [4]: 1) algorithms based on functional decomposition (see [17]–[20]); 2) algorithms based on explicit (see [9], [21]–[23]) or implicit network enumeration (see [5], [6], [24]–[28]); and 3) hybrid approaches that are based on both functional and structural properties (see [4], [29], [30]). Our approach is based on implicit enumeration. The associated techniques, besides hybrid approaches, is considered the most practical ones [4]. We only focus on algorithms from these two categories in the remaining discussion. Further information on the other algorithms is described in [4].

One of the first algorithms to find optimum circuit realizations with implicit enumeration was proposed by Muroga and Ibaraki [5]. They present a method based on integer linear programming. Although they put an emphasis on the synthesis of multilevel NOR networks, their approach is generic and can take into account several other network restrictions.

Ernst [4] has thoroughly investigated the problem of optimal multilevel logic synthesis. Her algorithm is implemented using a branch-and-bound method similar to the ones proposed by Davidson [29] and Culliney *et al.* [30]. The algorithm allows to handle different synthesis options and can be adapted to various cost criteria. Also, being based on branch-and-bound, the algorithm can easily be adapted to relax the optimality guarantee and find networks of near-optimal cost.

Knuth [6] has shown an SAT encoding to find size-optimum networks consisting of binary logic gates, borrowing ideas presented in [27] and [28]. His formulation requires the input function to be represented as a truth table and he shows how symmetry breaking and the restriction to normal functions can reduce the search space, borrowing previous ideas presented for explicit network enumeration [23].

The striking advantage of using an SAT or an SMT-based approach is that advances in new solvers directly influence the performance of our exact synthesis algorithm. Techniques used in branch-and-bound algorithms, such as presented in [4], mimic a lot of techniques that are implemented inside SAT and SMT solvers, however, tailored for the dedicated use of exact synthesis. This may be advantageous at the time the algorithm is implemented but not sustainable in the long run as SAT and SMT progress cannot affect such an application specific implementation.

III. Notation and Definitions

A. Boolean Functions

In this paper, we are concerned with Boolean functions $f : \mathbb{B}^n \to \mathbb{B}$ that map n input truth values to one output truth value. The central function in this paper is the *majority-of-three function*. The majority function 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 inputs are true. The majority function is monotone and self-dual [23] 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., $\langle 0xy \rangle = x \wedge y$ and $\langle 1xy \rangle = x \vee y$.

We like to emphasize some basic identities of the majority function which are essential for the forthcoming definition of MIGs. First, all three arguments to the majority function are commutative, i.e., $\langle xyz \rangle = \langle yxz \rangle = \langle zxy \rangle$. Also, the majority function evaluates to a single argument if two arguments are equal or inverse to each other, i.e., $\langle xxy \rangle = x$ and $\langle x\bar{x}y \rangle = y$. Since the majority function is self-dual, inverters can be *propagated* from the inputs to the outputs, i.e., $\langle \bar{x}\bar{y}\bar{z} \rangle = \overline{\langle xyz \rangle}$.

The majority function can be generalized for an odd number of n variables to $\langle x_1 \ldots x_n \rangle = [x_1 + \cdots + x_n > (n/2)]$. Many of the contributions in this paper can be generalized to this general majority function, although the description is restricted to 3-input majority functions in this paper. More details about general majority functions and their properties can be found in [31] and [32].

An interesting class of Boolean functions is the class of symmetric Boolean functions which occur frequently throughout this paper. Let $0 \leq i_1 < \cdots < i_k \leq n$ be a given sequence of distinct numbers ranging from 0 to n. Then

$$S_{i_1,\ldots,i_k}(x_1, \ldots, x_n) = [x_1 + \cdots + x_n \in \{i_1, \ldots, i_k\}] \tag{2}$$

is an n-variable symmetric function that evaluates to true if and only if the number of 1s in the input assignment equals a number in the given sequence. If the sequence of numbers is the interval $[0, i_k]$ or $[i_1, n]$, we can write $S_{\leq i_k}$ or $S_{\geq i_1}$, respectively. These special cases are also called *threshold functions* with unity weights.[1] Majority functions are a special case of threshold functions with unity weights and the general n-variable majority function is described by the symmetric function $S_{\geq \lceil n/2 \rceil}$.

B. Majority-Inverter Graphs

An MIG is a combinational logic network that has only majority-of-three and inverters as possible operations. Together with a constant value, it is a universal representation, as one can represent conjunction using the majority function, which forms a universal gate set together with inversion. Formally, an MIG of size r over variables $x_1, \ldots, x_n$ is a sequence $(x_{n+1}, \ldots, x_{n+r})$ of *gates* that combine previous gates using the majority function

$$x_i = \langle abc \rangle \text{ for } n < i \leq n + r. \tag{3}$$

Child gates are defined as $a = x_{s_{1i}}^{p_{1i}}$, $b = x_{s_{2i}}^{p_{2i}}$, and $c = x_{s_{3i}}^{p_{3i}}$, where $0 \leq s_{1i} < s_{2i} < s_{3i} < i$ are indexes pointing to the operands and $0 \leq p_{1i}, p_{2i}, p_{3i} \leq 1$ with $p_{1i} + p_{2i} + p_{3i} \geq 2$ are the operands' polarities. Note that the operands are ordered by their index and that at most one of the operands is inverted. This does not affect the universality of the representation due

[1] General threshold functions $f(x_1, \ldots, x_n) = [w_1x_1 + \cdots + w_nx_n \geq t]$ have integer coefficients w_i and an integer threshold value t (see [23]).

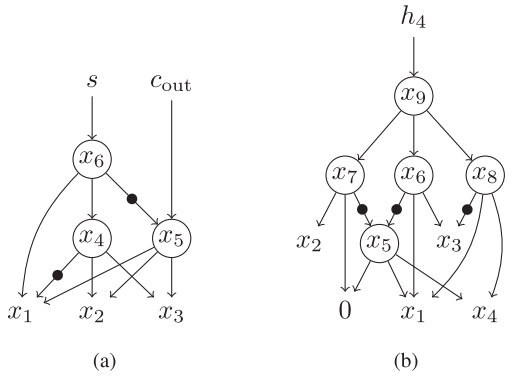


Fig. 1. MIG for (a) full adder and (b) hidden weighted bit function.

to commutativity and inverter propagation. We define $x_0 = 0$, such that we can realize AND and OR using MAJ. Finally, we define *outputs* $f_1, \ldots, f_m$ with

$$f_i = x_{s_i}^{p_i} \quad \text{for} 1 \leq i \leq m \tag{4}$$

where $0 \leq s_i \leq n + r$ and $0 \leq p_i \leq 1$.

The definition may seem quite involved at first sight, but in practical examples the sequences are fairly simple.

Example 1: The sequence $x_4 = \langle \bar{x}_1 x_2 x_3 \rangle$, $x_5 = \langle x_1 x_2 x_3 \rangle$, and $x_6 = \langle x_1 x_4 \bar{x}_5 \rangle$ computes the full adder with sum $s = x_6$ and carry-out $c_{\text{out}} = x_5$. The hidden weighted bit function [33] is defined as $h_n(x_1, \ldots, x_n) = x_{x_1 + \cdots + x_n}$ with $x_0 = 0$. The sequence $x_5 = \langle 0 \bar{x}_1 x_4 \rangle$, $x_6 = \langle x_1 x_3 \bar{x}_5 \rangle$, $x_7 = \langle 0 x_2 \bar{x}_5 \rangle$, $x_8 = \langle x_1 \bar{x}_3 x_4 \rangle$, and $x_9 = \langle x_6 x_7 x_8 \rangle$ computes $h_4(x_1, x_2, x_3, x_4) = x_9$.

The definition in (3) avoids some redundant representations. First, since the majority operation is commutative, an order is enforced on the operands. Second, we implicitly propagate inverters by allowing at most one operand of an MAJ operation to be complemented. Note that this has no effect on the universality and size of the representation, as inverters are simply moved toward the outputs.

It is possible to represent and illustrate the sequence as a directed graph with vertices for $x_0, \ldots, x_{n+r}$ and $f_1, \ldots, f_m$. The graph has edges from gates x_i to its operands $x_{s_{1i}}$, $x_{s_{2i}}$, and $x_{s_{3i}}$ for $n \leq i \leq n+r$. Also, we have edges from outputs f_i to gates or inputs x_{s_i} for $1 \leq i \leq m$. Polarities are represented using edge attributes: complements are indicated by a solid black circle on the edge. One readily sees that the resulting graph is acyclic.

Example 2: Fig. 1 shows the graph representations for the full adder and the hidden weighted bit function from Example 1. Note that the arrows of the edges are inverse to the direction of computation. We chose this way of visualization for a better understanding of the SMT encoding described in the following section.

For convenience, we introduce the notation $\text{children}(i) = \{s_{1i}, s_{2i}, s_{3i}\}$ and $\text{pol}(i) = (p_{1i} p_{2i} p_{3i})_2$ for $n < i \leq n + r$ to refer to the *operands* and *polarities* of a gate. A *path* is a sequence $i_1, i_2, \ldots, i_k$ such that:

1) $i_1 \leq n$, i.e., it starts in a constant or a primary input;
2) $i_j \in \text{children}(i_{j+1})$ for $1 \leq j < k$, i.e., it is connected via gates;
3) there exists an $l \in \{1, \ldots, m\}$ such that $s_l = i_k$, i.e., it ends in an output.

We refer to k as the *length* of the path. A longest path in an MIG is called a *critical path*. If k is the length of a critical path in an MIG, then the *depth* of the MIG is $k - 1$.

Example 3: The path $2, 4, 6$ in the full adder example is a critical path of length 3. The depth of the full adder is 2.

In order to compute the depth of an MIG we set levels $\ell_i = 0$ for $i \leq n$ and then compute levels ℓ_i for each gate $n < i \leq n + r$

$$\ell_i = \max\{\ell_{s_{1i}}, \ell_{s_{2i}}, \ell_{s_{3i}}\} + 1. \tag{5}$$

The depth of an MIG is then the maximum level over all outputs

$$\max_{f_i = x_{s_i}^{p_i}} \{\ell_{s_i}\}. \tag{6}$$

Two gates x_i and x_j are *structurally equivalent* if $\text{children}(i) = \text{children}(j)$, and $\text{pol}(i) = \text{pol}(j)$. The gates are *functionally equivalent* if x_i and x_j represent the same function. Structural equivalence implies functional equivalence and is easy to detect. We make use of structural hashing to ensure that no two gates in an MIG are structural equivalent. More advanced techniques [34] can be employed to guarantee *functionally reduced* MIGs in which gates that are functionally equivalent (up to complementation) are merged into *choice nodes* [35]. Multiple equivalent MIGs may be merged into a single functionally reduced MIGs in order to reduce structural bias in technology mapping [34], [36].

C. Cut Enumeration

A cut of a gate in an MIG is a subnetwork with a bounded number of inputs. In order to formally define cut enumeration, we need to define the notion of *gate paths*, denoted $\text{paths}(i)$, which are all paths that start from a primary input and terminate in gate x_i. Given an MIG of size r, we define $\text{paths}(i) = i$ for all $1 \leq i \leq n$ and

$$\text{paths}(x_i) = \bigcup_{\substack{j \in \text{children}(i) \\ j \neq 0}} \{\text{paths}(j), i\}. \tag{7}$$

Note that paths to constant zero are discarded. Let $p = i_1, \ldots, i_k$ be a path, then for convenience we can treat p as a set $\{i_1, \ldots, i_k\}$ when being applied to set operations.

Given an MIG of size r, a pair (i, L) consisting of a *root* $1 \leq i \leq n + r$ and *leafs* $L \subseteq \{1, \ldots, n + r\}$ is called a *cut* (see [37]), if:

1) $p \cap L \neq \emptyset$ for all $p \in \text{paths}(i)$, i.e., all paths to x_i contain at least one leaf $l \in L$;
2) and for all $l \in L$ there exists $p \in \text{paths}(i)$ with $l \in p$, i.e., each leaf is contained in at least one path.

A cut is called k-feasible, if $|L| \leq k$ and we denote all k-feasible cuts of a gate x_i as

$$\text{cuts}_k(i) = \{L \mid (i, L) \text{ is a cut and } |L| \leq k\}. \tag{8}$$

Each cut (i, L) describes a subgraph which may have gates apart from x_i and the ones in L. These gates are called *internal gates*. All k-feasible cuts can be generated using the recursive algorithm

$$\text{cuts}_k(i) = \begin{cases} \{\{\}\} & \text{if } i = 0 \\ \{\{i\}\} & \text{if } 1 \leq i \leq n \\ \text{cuts}_k(s_{i1}) \otimes_k \text{cuts}_k(s_{i2}) \otimes_k \text{cuts}_k(s_{i3}) & \text{otherwise.} \end{cases}$$

The operation

$$M_1 \otimes_k M_2 = \{m_1 \cup m_2 \mid m_1 \in M_1, m_2 \in M_2, |m_1 \cup m_2| \leq k\}$$

is a saturating union over all combinations of subsets. An exhaustive enumeration of all cuts in an MIG is feasible as long as $k \leq 6$ [38]. For k larger than 6, priority cuts provide an alternative approach [38].

D. NPN Classification

Two functions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ are NPN-equivalent, if there exists a permutation $\sigma \in S_n$ (with S_n being the symmetric group over $\{1, \ldots, n\}$) and polarities $p, p_1, \ldots, p_n \in \mathbb{B}$ such that $f(x_1, \ldots, x_n) = g^p(x_{\sigma(1)}^{p_1}, \ldots, x_{\sigma(n)}^{p_n})$, i.e., g can be made equivalent to f by negating inputs, permuting inputs, or negating the output. NPN-equivalence is an equivalence relation that partitions the set of all Boolean functions over n variables into a smaller set of NPN classes. As an example all 2^{2^n} Boolean functions over n variables can be partitioned into 2, 4, 14, 222, 616 126 NPN classes for $n = 1, 2, 3, 4, 5$. As the representative of each NPN class, we take the function with the smallest truth table, when truth tables are viewed as a binary number of 2^n bits. For a detailed introduction into NPN classification the reader is referred to [39] and [40].

IV. Exact Synthesis

This section describes the exact synthesis formulation as SMT problem in detail. After the next section provides some general introductory comments, the subsequent sections deal with the encoding of the variables and constraints in the SMT formulation.

A. Generalities

Exact synthesis describes the optimization problem of finding the optimum MIG $x_{n+1}, \ldots, x_{n+r}$ that realizes a given function $f : \mathbb{B}^n \to \mathbb{B}$, i.e., $f = x_{n+r}^p$ for some p. As cost criteria we consider in this paper MIGs with:

1) the smallest size r (referred to as *size-optimum*);
2) the smallest depth within the MIGs of smallest size (referred to as *size/depth-optimum*);
3) the smallest size within the MIGs of smallest depth (referred to as *depth/size-optimum*).

The second and third criteria have both size and depth as objectives but in alternating roles for the primary and secondary objective. As an example, in *size/depth-optimum* MIGs, the size is the primary objective.

The large amount of different MIG structures rule out direct algorithms to solve this optimization problem. Instead, we propose to solve the optimization problem by solving a sequence of decision problems (see also [41]). The decision problem asks whether there exists an MIG of a given size r and a given maximum depth d that realizes f. Then, the decision problem is

$$\exists x_{n+1}, \ldots, x_{n+r}\ \exists p : \left(x_{n+r}^p = f\right) \wedge (\ell_{n+r} \leq d). \tag{9}$$

Note that ℓ_r corresponds to the depth of the overall MIG, as it has only one output.

We refer to the decision problem in (9) as HasMIG(f, r, d) and it either returns an MIG $x_{n+1}, \ldots, x_{n+r}$ and output polarity p, if one exists, or *false* otherwise.

For size-optimum MIGs the depth is irrelevant. Then, the clause $(\ell_{n+r} \leq d)$ is removed from (9) and the decision problem is referred to as HasMIG(f, r).

The decision problem can be solved using an SMT solver, if we find an encoding for the existentially quantified variables and for the equivalence and depth constraint on the right-hand side of (9). The remainder of this section describes the encoding of the variables, which are essentially the operands s_{1i}, s_{2i}, s_{3i} and the polarities p_{1i}, p_{2i}, p_{3i}, and the encoding of the constraints.

We will use a simple preprocess to check whether f can be represented without any gate, i.e., HasMIG$(f, 0)$ is satisfiable. This is true, if and only if f is constant or $f = x_i^p$ for some i and some p. We do not need to utilize an SMT solver for such a simple check and therefore assume that $r > 0$ when describing the encoding for the SMT solver.

B. Encoding of Variables

According to the definition of MIGs we introduce integer variables s_{1i}, s_{2i}, s_{3i} and Boolean variables p_{1i}, p_{2i}, p_{3i} for each $n < i \leq n + r$ and one other Boolean variable p for the output polarity. Also, to follow the definition we need to add constraints to ensure: 1) that there are no cycles; 2) that the children are ordered; and 3) that at most one child is complemented.

Cycles are avoided by adding the constraint

$$s_{ci} < i \qquad \text{for } 1 \leq c \leq 3. \tag{10}$$

The constraint

$$(s_{1i} < s_{2i}) \wedge (s_{2i} < s_{3i}) \tag{11}$$

ensures that the children are ordered. Finally

$$(p_{1i} \vee p_{2i})(p_{1i} \vee p_{3i})(p_{2i} \vee p_{3i}) \tag{12}$$

ensures that at most one children is complemented. Note that (12) is the conjunctive normal form of the majority operation (1).

C. Encoding of the Equivalence Constraint

We propose two ways to encode the equivalence constraint depending on the representation of f, which can be either *explicit* in terms of a truth table or *symbolic*, e.g., in terms of a Boolean expression or a logic circuit, e.g., a nonoptimized MIG.

1) Explicit Function Representation: In this case f is represented as a truth table with truth values for all 2^n input assignments. The idea is to "copy" each majority gate with index $i \in \{n + 1, \ldots, n + r\}$ for each input assignment $t \in \{0, \ldots, 2^n - 1\}$. Each copy represents the simulation of one input assignment. The copies are "connected" by the s_{ci} and p_{ci} (for $1 \leq c \leq 3$) variables which have the same values in each copy, since they represent the structure of the MIG. For each gate x_i in each copy t four Boolean variables $a_{1i}^{(t)}$, $a_{2i}^{(t)}$, $a_{3i}^{(t)}$, and $b_i^{(t)}$ encode the simulation values of the input assignment in the MIG. The variables $a_{ci}^{(t)}$ represent the value at input c of gate x_i and the variable $b_i^{(t)}$ represents the value at the output of gate x_i.

We need to add constraints that ensure: 1) the semantics of the majority functionality of each gate; 2) the propagation of simulation values through the circuit; and 3) the correct function value at the root gate k.

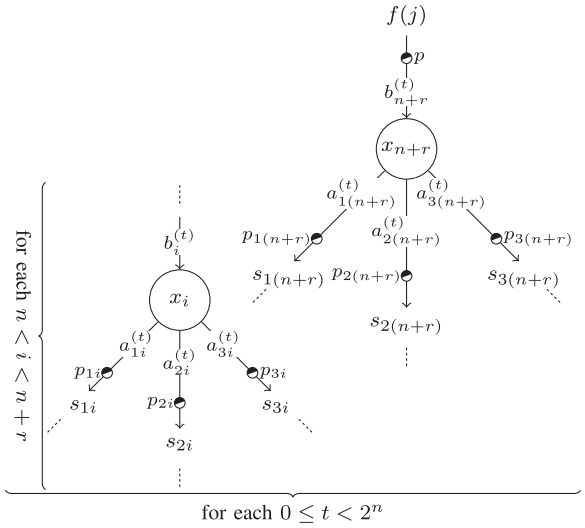


Fig. 2. Explicit function representation.

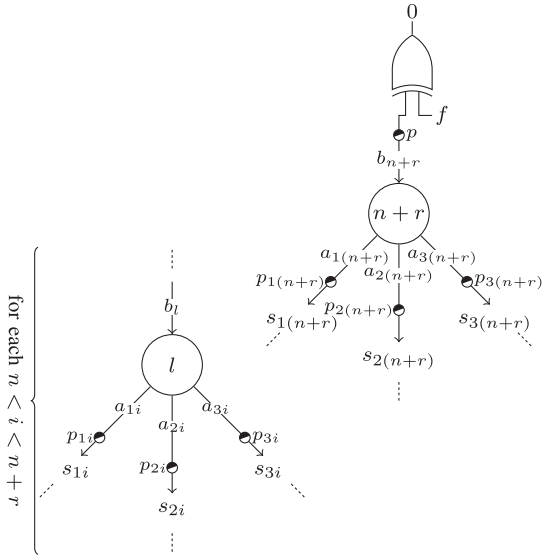


Fig. 3. Symbolic function representation.

a) Majority functionality: The formula

$$b_i^{(t)} = \langle a_{1i}^{(t)} a_{2i}^{(t)} a_{3i}^{(t)} \rangle \tag{13}$$

ensures the correct functionality of the gate, i.e., the output value $b_i^{(t)}$ of the gate x_i is the majority of the gate's three input values $a_{1i}^{(t)}$, $a_{2i}^{(t)}$, and $a_{3i}^{(t)}$ for all assignments t.

b) Input connections: Constraints on the input connections are given in terms of implications of the variables s_{ci}, where c ranges from 1 to 3. The constraint

$$(s_{ci} = j) \rightarrow \left(a_{ci}^{(t)} = b_j^{(t)} \oplus \bar{p}_{ci}\right) \tag{14}$$

ensures that values are propagated correctly through the MIG that is represented by the s- and p-variables. In (14), j ranges from 0 to $i-1$ and we define $b_0^{(t)} = 0$ and $b_j^{(t)}$ is the t^{th} bit in the truth table representation of x_j when $1 \leq j \leq n$.

c) Function semantics: Finally, the correct function semantics is ensured by the formula

$$b_{n+r}^{(t)} = \bar{p} \oplus f(t). \tag{15}$$

Note that $n + r$ is the largest node index and therefore refers to the root node.

The encoding is illustrated by means of Fig. 2. The "$\circ$" together with the polarity variable p next to it represent a conditional edge complement, which is assumed to exist if $p = 0$. Note that each node x_i is copied $r-1$ times; the output node x_{n+r} is shown explicitly, since its output value $b_{n+r}^{(t)}$ in polarity p must be equal to $f(t)$. Hence, all r nodes are copied 2^n times for each entry t in the truth table. Also note that node x_i can have more than one ingoing edge, if more than one parent points to it.

2) Symbolic Function Representation: In case f is represented symbolically, the equivalence constraint is encoded by universally quantifying over the input variables

$$\forall x_1, \ldots, x_n : (b_{n+r} \oplus p) = f. \tag{16}$$

In (16) f is assumed to be the function value of f through some encoding, e.g., Tseytin encoding [42] or EMS encoding [43]. The expression b_{n+r} represents the output value of the root gate x_{n+r}. The output expressions b_i are

$$b_i = \langle a_{1i} a_{2i} a_{3i} \rangle \tag{17}$$

with input expressions

$$a_{ci} = \bigwedge_{j=0}^{i-1} \left((s_{ci} = j) \rightarrow \bar{p}_{cl} \oplus b_j\right) \tag{18}$$

for all $n < i \leq n + r$ and $1 \leq c \leq 3$. The value of a_{ci} depends on the variables s_{ci} and p_{ci}. Variable s_{ci} determines the value of the input depending on where it points to. As in the explicit encoding we have $b_0 = 0$ and $b_j = x_j$ for $1 \leq j \leq n$. The variable p_{ci} determines the polarity of that value. SMT solvers can use tactics such as quantifier elimination in order to handle the universal quantifiers in (16).

In contrast to the encoding where f is explicitly represented, the variables a_{ci} and b_i variables are not existentially quantified since they depend on the values of x_i. Since b_i depends on the a_{ci} variables and the a_{ci} variables in turn depend on variables $b_{i'}$ with $i' < i$, the expressions in (17) and (18) result in a large nested "*let*" expression.

The encoding for a symbolic function representation is illustrated by means of Fig. 3. In contrast to the encoding for the explicit function representation, the variables are not copied for each input assignment. The XOR gate that connects the output value of the root node x_{n+r} with the output of the encoded f is set to 0 at its output, what ensures the function semantics for the MIG.

3) Soundness: Soundness follows immediately from the construction of the SMT formula as we reuse the exact same variables appearing in the definition of an MIG as variables in the SMT formula. We add a- and b-variables to encode the evaluation of the function f. The b values are related to the a-variables of the same gate via the majority operation and the a values are related to b-variables of lower gates via corresponding s- and p-variables which encode the structure of the MIG.

D. Encoding of the Depth Constraint

When considering depth constraints we are interested in finding an MIG which depth does not exceed a given value d, or in other words the level ℓ_{n+r} of the output gate does not exceed d. We add four additional integer variables ℓ_{1i}, ℓ_{2i}, ℓ_{3i},

Algorithm 1: Exact Synthesis for Size-Optimum MIGs

```
   Input  : Function f : B^n → B
   Output : Size-optimum MIG (x_{n+1}, ..., x_{n+r}), p
1  set r ← 0;
2  while true do
3  |  if HasMIG(f, r) then
4  |  |  return (x_{n+1}, ..., x_{n+r}), p
5  |  else
6  |  |  set r ← r + 1;
7  |  end
8  end
```

and ℓ_i for each gate $n < i \leq n + r$ to the SMT formulation, where ℓ_{ci} encodes the level of input c at gate x_i and ℓ_i encodes the depth at the output of gate x_i as in (5). Three types of constraints are required to ensure a correct interpretation. As described above, the case $r = 0$ is checked explicitly, and therefore we can assume that $d > 0$ when encoding the SMT formula.

a) Gate depth: The formula

$$\ell_i = \max\{\ell_{1i}, \ell_{2i}, \ell_{3i}\} + 1 \tag{19}$$

constrains that each gate's level is the maximum level of its children incremented by 1.

b) Depth propagation: The constraints

$$(s_{ci} = j) \rightarrow \left(\ell_{ci} = \ell_j\right) \qquad \text{for } n < i \leq n + r \tag{20}$$

where $\ell_j = 0$ for $j \leq n$ ensure that the depth values are correctly propagated through the circuit, similar to how the constraints (14) ensure a correct propagation of simulation values.

c) Depth bound: Finally, the formula

$$\ell_{n+r} \leq d \tag{21}$$

restricts the maximum depth, i.e., the depth of the output gate x_{n+r}, to the given bound d.

V. Synthesis Algorithms

This section describes how to utilize the decision problems HasMIG(f, k) and HasMIG(f, k, d) in order to find optimum MIGs with respect to the three cost criteria introduced in the previous section.

A. Size-Optimum Synthesis

Algorithm 1 describes the simplest algorithm that guarantees size-optimum results. Input is a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ and it returns a size-optimum MIG. The call to HasMIG(f, r) in line 3 is either unsatisfiable or returns a solution that is extracted from the satisfying assignment to the existentially quantified variables. Note that $x_{n+i} = \langle x_{s_{1i}}^{p_{1i}} x_{s_{2i}}^{p_{2i}} x_{s_{3i}}^{p_{3i}} \rangle$ is composed of six variables in the SMT instance. Also the output polarity p is returned. Then $f = x_{n+r}^{p}$, after the algorithm terminated. In order to guarantee a size-optimum result r is initially set to 0 and incremented as long as no MIG of size r can be found.

Instead of a linear search for an optimum size r, the algorithm can also be implemented using binary search. This is enabled due to the following lemma, which states that HasMIG(f, r) has a monotonic behavior.

Algorithm 2: Exact Synthesis for Size/Depth-Optimum MIGs

```
   Input  : Function f : B^n → B
   Output : Size/depth-optimum MIG (x_{n+1}, ..., x_{n+r}), p
1  set r ← 0;
2  while true do
3  |  if HasMIG(f, r) then
4  |  |  set d ← 0;
5  |  |  while true do
6  |  |  |  if HasMIG(f, r, d) then
7  |  |  |  |  return (x_{n+1}, ..., x_{n+r}), p
8  |  |  |  else
9  |  |  |  |  set d ← d + 1;
10 |  |  |  end
11 |  |  end
12 |  else
13 |  |  set r ← r + 1;
14 |  end
15 end
```

Lemma 1: If HasMIG(f, r) is satisfiable, then also HasMIG$(f, r + 1)$ is satisfiable.

Proof: Let $x_{n+1}, x_{n+2}, \ldots, x_{n+r}$ with $f = x_{n+r}^{p}$ for some p be a solution to HasMIG(f, r). Then $x'_{n+1} = x_{n+1}, \ldots, x'_{n+r-1} = x_{n+r-1}, x'_{n+r}, x'_{n+r+1} = x_{n+r}$ with $f = (x'_{n+r+1})^p$ is a solution to HasMIG$(f, r + 1)$ where $x'_{n+r} \neq x'_{n+i}$ for $i \in \{1, \ldots, r - 1, r + 1\}$ can be chosen arbitrarily.

The gate x'_{n+r} then has no parent and also no output points to it. Since all children of a majority gate need to be different according to the definition of MIGs, we cannot simply add a gate $x_{n+r+1} = \langle x_{n+r} x_{n+r} x_{n+r} \rangle$. ∎

Lemma 1 shows that there is a clear cut between unsatisfiable and satisfiable instances of HasMIG(f, r).

However, binary search may not necessarily perform faster since the runtime of the HasMIG(f, r) calls can diverge significantly for different values of r; especially since unsatisfiable instances of HasMIG(f, r) are typically harder to solve than satisfiable ones. In theory, the upper bound for the number of gates for arbitrary Boolean functions is $(2^n/n)(1 + 3(\lg n/n) + O((1/n)))$ given from Lupanov for circuits consisting of binary operations (see [23], [44]). However, for practical examples this upper bound is far too large and it is easier to derive an upper bound from heuristic synthesis methods. In our experimental environment we use the commands "*read_truth*; *strash*; *dc2*" in ABC [45] to first convert a truth table into an AIG and then optimize it for size. Since an AIG can be represented by an MIG with the same number of gates, it can serve as an upper bound. Instead of implementing a binary search, one can also reverse Algorithm 1, set r to an upper bound and then decrease until HasMIG(f, r) becomes unsatisfiable (see also [46]).

B. Size/Depth-Optimum Synthesis

The size-optimum MIG representation is not unique. There are possibly many MIGs that have the same size, but not the same depth. For example, consider the two MIGs in Fig. 4 that both realize the function $x_1 \oplus x_2 \oplus x_3$ with three gates. The second MIG in Fig. 4 has a depth of 2, which is optimum, while the first MIG has a depth of 3.

Algorithm 2 shows an extension of the size-optimum algorithm to find MIGs that have smallest area as primary cost

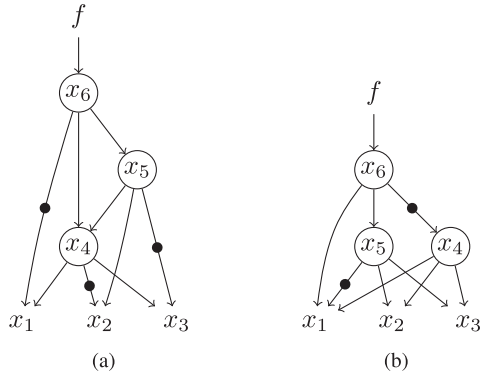


Fig. 4. Two MIGs for $f = x_1 \oplus x_2 \oplus x_3$ with 3 gates: (a) size-optimum (depth 3) and (b) size/depth-optimum (depth 2).

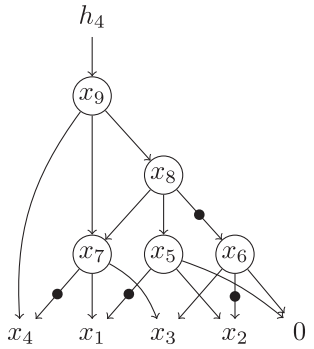


Fig. 5. Alternative size/depth-optimum MIG for h_4.

criteria and the smallest depth as secondary cost criteria. We first determine the minimum number of gates r with the approach described in the previous section. We then fix r and increase the depth d until a satisfying solution is found using the extension described in this section.

One can obtain the algorithm by replacing line 4 in Algorithm 1 by a loop that determines the best depth. Instead of starting the loop from $d = 0$, one can set $d \leftarrow \ell_{n+r}$ in line 4, i.e., to the level or the root gate x_{n+r} extracted from the satisfiable solution. Then d is decreased until HasMIG(f, r, d) becomes unsatisfiable and the previously found solution is returned.

Global Delay-Aware Synthesis: In practical applications the functions to be synthesized are often extracted from larger circuits. The optimized subnetwork can be significantly different from the starting point as no structural information is taken into consideration. When optimizing for size as a first criteria, it is guaranteed that the overall network does not increase in size after replacing the subnetwork by its optimum counterpart. However, the replacement may have a negative effect on the overall delay, even when applying size/depth-optimum synthesis.

We illustrate this effect with an example. Consider the MIG for the h_4 in Fig. 1(b), which is size/depth-optimum with a size of 5 and a depth of 3. The longest paths from the output gate x_9 to x_2 and x_3 have length of 2, while the longest paths to the other two inputs have length of 3. Now, assume that this network is a subnetwork extracted from a larger MIG in which the path from x_2 to x_9 is on the critical path. Size/depth-optimum synthesis may as well have returned the solution illustrated in Fig. 5 which also has a size of 5 and depth of 3. However, the longest path from the x_9 to all inputs is 3. Hence, replacing the MIG of Fig. 1(b) with the MIG of Fig. 5 in the context of the larger MIG will increase the critical path by 2.

Algorithm 3: Exact Synthesis for Depth/Size-Optimum MIGs

```
Input  : Function f : B^n → B
Output : Depth/size-optimum MIG (x_{n+1}, ..., x_{n+r}), p
1 set d ← 0;
2 while true do
3 |   foreach r ∈ {0, ..., (3^d − 1)/2} do
4 |   |   if M ← HasMIG(f, r, d) then
5 |   |   |   return (x_{n+1}, ..., x_{n+r}), p
6 |   |
7 |   end
8 |   set d ← d + 1;
9 end
```

The problem can be circumvented by considering the depth information of each node in the original circuit in the encoding of the synthesis problem. Instead of setting $\ell_j = 0$ for $j \leq n$ in (20) we preassign these values with the depth information from the larger circuit.

C. Depth/Size-Optimum Synthesis

When considering depth as the primary cost criteria, the size of the optimum MIG may be larger than the size of the smallest MIG (when depth is not considered). As a result, we need to perform the algorithm in a different manner if we want to find MIGs with the smallest depth. Algorithm 3 illustrates the procedure. We start by setting $d \leftarrow 0$ and then increase the number of gates k. If no solution can be found, d is increased, otherwise the optimum has been determined. We know when to increase d, because the number of gates in an MIG with depth d is bounded by

$$3^0 + 3^1 + \cdots + 3^{d-1} = (3^d - 1)/2 \tag{22}$$

gates. This is the case, when no sharing is possible and the MIG is a tree.

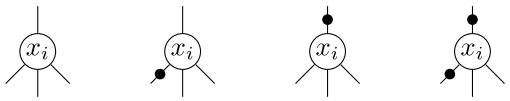
One can strengthen this upper bound by taking further considerations into account. For example, the first level of MIG nodes (closest to the inputs) is bounded by $4 \cdot \binom{n}{3}$: for each triple of three inputs we can have eight different majority functions and the complemented edges allow us to represent them by four different gates. Hence, the last term in (22) can be replaced by $\min\{3^{d-1}, 4 \cdot \binom{n}{3}\}$. One can imagine similar considerations for the other levels, however, we do not investigate this further in the scope of this paper.

VI. Optimizations

This section shows different optimizations that speed up the solving time. First, we show that it is possible to discard the output polarity variable when considering only normal functions for synthesis. We introduce several symmetry breaking rules which discard equivalent solutions and thereby reduce the solving time. Finally, we show that the encoding of integers has an effect and propose how the exact algorithm can be turned into a heuristic.

A. Normal Functions

Knuth [6], [23] restricted both explicit and implicit enumeration to *normal* functions and can thereby cut the search space in half. A Boolean function f is normal, if $f(0, \ldots, 0) = 0$. A normal function can be represented by a logic network that consists only of normal gates, i.e., gates which represent normal functions. This also applies to optimum networks. There are four possible occurrences of gates in an MIG according to its definition, which are



(23)

modulo a permutation of the children (i.e., the input complement may be on any of the three children). The last two cases only occur due to the possibility of complementing the output functions, and these two cases are the only gates which are not normal, i.e., $f(0, 0, 0) = 1$.

We can exploit this property and restrict the synthesis algorithm to only consider normal functions. This simplifies the decision problem as the variable p can be dropped from the formula, reducing (9) to

$$\exists x_{n+1}, \ldots, x_{n+r} : \left(x_{n+r} = f'\right) \wedge (\ell_{n+r} \leq d) \tag{24}$$

where $f' = f$, if f is normal, and $f' = \bar{f}$, otherwise. In the latter case, the outgoing edge of the root gate is complemented.

B. Symmetry Breaking

All the constraints described in Section IV ensure a correct result in case of a satisfying assignment. In order to reduce the search space, we make use of symmetry breaking which discards equivalent solution of the same quality. In order to demonstrate the effect of symmetry breaking, we introduce a running example in this section. We apply different configuration of symmetry breaking rules to the function $S_{1,3,4} = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \vee x_1x_2x_3x_4$, which is a worst-case example of a 4-variable function. The optimum MIG requires seven gates. Further details are given in the experimental evaluation. In the running experiment, we measure the runtime of finding the optimum MIG under the consideration of different symmetry breaking rules.

1) Breaking Using Structural Hashing: In an optimum MIG no gate can occur twice. This can be enforced by adding the clauses

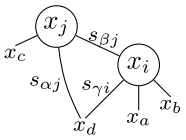
$$\bigwedge_{n<i<j\leq n+r} \left(\left(s_{1i} \neq s_{1j}\right) \vee \left(s_{2i} \neq s_{2j}\right) \vee \left(s_{3i} \neq s_{3j}\right) \vee \left(p_{1i} \neq p_{1j}\right) \vee \left(p_{2i} \neq p_{2j}\right) \vee \left(p_{3i} \neq p_{3j}\right)\right). \tag{25}$$

These constraints have no effect on the number of satisfiable solutions, but can reduce the runtime. In the running example (see Fig. 6) the runtime is almost halved, and a solution is found after 123.13 instead of 217.13 s.

2) Breaking Using Associativity: For Boolean functions u, x, y, z, the associativity law in majority logic is

$$\langle xu\langle yuz\rangle\rangle = \langle yu\langle xuz\rangle\rangle = \langle zu\langle xuy\rangle\rangle. \tag{26}$$

In a size-optimum MIG we only need to consider one of these three cases. However, the majority operations in (26) are only special cases of a more general rule, since the common child u may be at different positions. The general gate structure that enables this associativity property is



(27)

In other words, x_j points to x_i with child β, i.e., $s_{\beta j} = i$. Child α of x_j and child γ of x_i point to the same gate x_d, i.e., $s_{\alpha j} = s_{\gamma i} = d$. On gate x_j, the child which is not α and β points to x_c, i.e., $s_{(6-\alpha-\beta)j} = c$. Further, we want that the most right child of x_i that is not γ point to x_b, i.e., $s_{(5-\max(2,\gamma))i} = b$. In order to break symmetries for such structures, we enforce that $a \leq b \leq c$. Note that $a < b$ is already guaranteed due to (11) in the definition of MIGs, and therefore it suffices to enforce $b \leq c$. From this observation, we can derive the constraint

$$\bigwedge_{n<i<j\leq n+r} \bigwedge_{\substack{1\leq\alpha,\beta,\gamma\leq 3\\ \alpha\neq\beta}} \left(s_{\beta j} = i \wedge s_{\alpha j} = s_{\gamma i} \wedge p_{\alpha j} = p_{\gamma i}\right) \rightarrow \left(s_{(5-\max(2,\gamma))i} \leq s_{(6-\alpha-\beta)j}\right). \tag{28}$$

Symmetry breaking based on the associativity law leads to further runtime improvement. In the running example when being applied together with breaking based on structural hashing, the runtime is reduced from 123.13 s to 38.98 s.

Notice that this symmetry breaking cannot be used when optimizing for depth, as the levels of x_a, x_b, and x_c in (27) are different and reordering may lead to a different overall depth of the MIG.

3) Breaking Using Colexicographic Ordering: If a gate x_i is not a child of the successor x_{i+1}, then they can be executed in any order, e.g., the MIG will not change if we replace $x_7 = \langle x_3x_5x_6\rangle$, $x_8 = \langle x_2x_4x_5\rangle$ by $x_7 = \langle x_2x_4x_5\rangle$, $x_8 = \langle x_3x_5x_6\rangle$. In order to disambiguate this case, we enforce a colexicographic order on the children, i.e., only the second order in the example will be valid, since $2 < 3$. Notice that x_i is a child of x_{i+1},

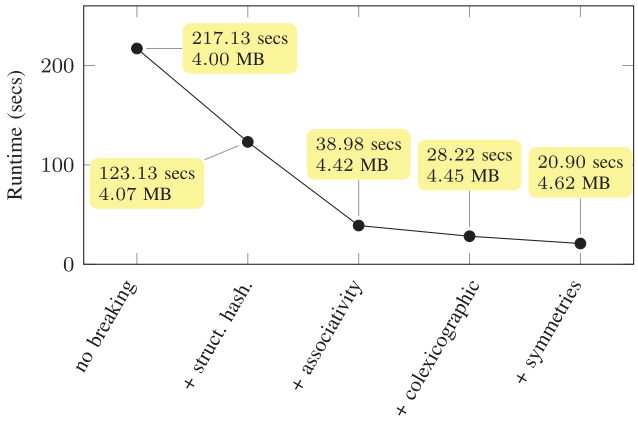


Fig. 6. Runtime and memory usage to find optimum MIG for $S_{2,4}$ using different symmetry breaking rules. Each rule is applied additive with respect to the previous ones, i.e., the rightmost entry considers the combination of all five symmetry breaking rules.

if and only if $s_{3(i+1)} = i$ due to (11). The constraint to break symmetries is

$$\bigwedge_{n<i<n+r} \big(s_{3(i+1)} = i\big) \rightarrow \Big(\big(s_{1i} < s_{1(i+1)}\big) \vee \big(\big(s_{1i} = s_{1(i+1)}\big) \wedge \big(s_{2i} < s_{2(i+1)}\big)\big) \vee \big(\big(s_{1i} = s_{1(i+1)}\big) \wedge \big(s_{2i} = s_{2(i+1)}\big) \wedge \big(s_{3i} \leq s_{3(i+1)}\big)\big)\Big). \quad (29)$$

It is important to use "$\leq$" for the comparison between s_{3i} and $s_{3(i+1)}$ because it is possible that all children of x_i and x_{i+1} are the same but with different polarities.

In the running example, we can improve the runtime by another 10 s from 38.98 s to 28.22 s when being applied to all previously introduced symmetry breaking rules.

The symmetry breakers that have been discussed so far can be applied in general to all functions that are input to the exact synthesis algorithm. The next two symmetry breakers take properties of the input function into account. Consequently, a preprocess is required to extract these properties. For the proposed symmetry breakers, the runtime to obtain the properties is negligible.

4) Breaking Using Support Information: Let $f : \mathbb{B}^n \rightarrow \mathbb{B}$ be the function to be synthesized. If f does not depend on x_j with $1 \leq j \leq n$, i.e., a change in x_j does not change the output value of f, then no gate points to x_j in an optimum MIG for f. We refer to the *functional support* of f as the set of variables on which f depends. It is unlikely that input variables are not in the functional support for practical examples. However, for small functions the functional support can be easily computed. For each x_j that is not in the functional support, the constraint

$$\bigwedge_{n<i\leq n+r} (s_{1i} \neq j) \wedge (s_{2i} \neq j) \wedge (s_{3i} \neq j) \quad (30)$$

is added. This symmetry breaking constraint will not lead to fewer satisfiable solutions but can reduce the runtime.

We have excluded this symmetry breaking optimization in our experiment in Fig. 6, because the considered function depends on all variables.

5) Breaking Using Symmetric Variables: A function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ is symmetric in two different variables x_k and x_l if

$$f(x_1, \ldots, x_k, \ldots, x_l, \ldots, x_n) = f(x_1, \ldots, x_l, \ldots, x_k, \ldots, x_n) \quad (31)$$

that is interchanging x_k and x_l does not change the output values of f. Let us assume that $k < l$. Then, we can break the symmetry by enforcing that x_k's first appearance in the MIG must be before x_l's first appearance. This can be expressed using the following constraint:

$$\bigwedge_{n<i\leq n+r} \bigwedge_{1\leq c\leq 3} \left((s_{ci} = k) \wedge \bigwedge_{(d,j)\prec(c,i)} \big(s_{dj} \neq k\big) \right) \rightarrow \bigwedge_{(d,j)\prec(c,i)} \big(s_{dj} \neq l\big) \quad (32)$$

where $(d, j) \prec (c, i) \Leftrightarrow (d \leq 3) \wedge (j \leq i) \wedge ((j < i) \vee (d < c))$. The constraint considers cases in which $s_{ci} = k$, i.e., the cth child of gate x_i points to x_k and no gate and no child before points to x_k. Then, no child of a previous gate can point to x_l, but also no smaller child of gate x_i can point to x_l.

Algorithm 4: Heuristic Synthesis for Size-Optimized MIGs

Input : Function $f : \mathbb{B}^n \rightarrow \mathbb{B}$
Output : Size-optimized MIG $(x_{n+1}, \ldots, x_{n+r}), p$

```
set r ← 0;
while true do
    if HasMIG_TO(f, r, t(r)) then
        return (x_{n+1}, ..., x_{n+r}), p
    else
        set r ← r + 1;
    end
end
```

This symmetry breaking rule has even further impact on the experiment function in Fig. 6. The runtime can be reduced to 20.90 s. That is, the overall improvement when applying all symmetry breaking rules is more than 10×.

C. Encoding of Integers

The s-variables in the encoding are integers that point to preceding gates, input variables, or constants. The introduced constraints only require comparison operators and no arithmetic operations on these variables. SMT solvers allow several possibilities to encode integers, e.g., as bitvectors of fixed size or as integer types. The choice of encoding has an impact on the runtime.

As an example, when enabling all symmetry breaking rules for the function $S_{2,4}$, which has been used as running example in the previous section, the runtime increases from 20.90 s when using the bitvector encoding to 56.71 s when using an integer encoding. We have observed this trend in all considered benchmarks.

D. Timeout Heuristic

The search for a size-optimum MIG with r gates requires to solve r decision problems using an SMT solver (recall that HasMIG$(f, 0)$ is checked explicitly). Most of the overall runtime in this case is typically required to prove that HasMIG$(f, r-1)$ is unsatisfiable, and often HasMIG(f, r) can be solved quickly as it stops once the first solution has been found.

We propose a heuristic that exploits this behavior. Algorithm 4 demonstrates the idea. Instead of HasMIG(f, r) the algorithm calls an alternative function HasMIG$_{\text{TO}}(f, r, t)$ which terminates with no result after t seconds. In this case, the algorithm behaves as if HasMIG(f, r) returns *unsatisfiable* and increments r by 1. Once r is large enough that the problem is satisfiable, a solution may be found within the time limit. However, if the timeout (TO) is chosen too small, it may not be sufficient to find a satisfying solution. In this case, no call to HasMIG$_{\text{TO}}(f, r, t)$ returns a solution and the algorithm is stuck in an infinite loop. A better strategy is to use a function $t(r)$ that returns a TO value depending on the progress in the algorithm. Ideally, the TO value increases proportional to r.

When using the TO heuristic with $t(r) = 1$ for the running example $S_{1,3,4}$ we can find the optimum MIG with seven gates in 2.65 s instead of 7.14 s.

Other functional symmetry breakers are possible but have not been evaluated in the scope of this paper. One can for example break symmetries based on unateness properties or functional decomposibility.

VII. Applications

Although the proposed exact synthesis algorithm is efficient for small functions, it does not scale for large functions. Further, it is not expected to find optimum networks for large functions. However, the exact synthesis algorithm can be used for large functions when being applied to small subnetworks to guarantee local optimality. In this section, we describe several application scenarios in which the exact synthesis algorithm can be employed. The description is kept brief. The applications have already been presented in the literature and references are provided that point to further details.

A. Functional Hashing

Functional hashing using exact synthesis has been proposed in [47]. The idea is to replace cuts by optimum MIGs. The algorithm is fast when the cut size is 4 or less, because, in this case, the optimum networks for all NPN classes of functions with up to four variables can be precomputed. The quality of results is affected by the strategy on how cuts are replaced. A *top-down* approach starts with cuts from the outputs and selects the cut with the largest improvement in gate reduction. Then, the algorithm recurs at the children of the replaced network and internal gates in a replaced cut are not further considered. A *bottom-up* approach computes the optimum replacement for *each* gate and then assembles the optimal network using a dynamic programming approach.

When replacing cuts, it is important that no internal gate fans out to other gates outside of the cut, because such internal gates do not exist anymore after the replacement. Two approaches were presented to counteract this problem. First, one can discard cuts with internal gates that fan out to other gates. Second, one can restrict replacement to fanout-free regions.

While this approach is successful in reducing the size of the overall network, the depth may significantly increase because critical paths can be extended with the replaced optimum networks (see previous discussion on global depth-aware synthesis, Section V-B). Heuristics can be used to circumvent this effect, however, this leads to less flexibilities in reducing size.

B. LUT-Based Optimization

In order to keep the depth of the circuit in a reasonable boundary when replacing subnetworks by optimum networks, lookup table (LUT)-based optimization provides an attractive alternative to functional hashing. The idea is to identity the subnetworks using LUT mapping. First the network is mapped into k-LUTs, e.g., in a depth-optimal manner [48]. Each k-LUT represents a k-variable Boolean function which is then used as input for exact synthesis. LUT-based mapping for exact synthesis has been proposed in [49].

Another difference of this approach compared to functional hashing is that it is not *in-place*. This means that it is essential to find an MIG for every subnetwork in order to derive an MIG globally for the whole function. In contrast, if a subnetwork cannot be optimized in functional hashing, the overall network is locally nonoptimal for this subnetwork, but the overall representation stays an MIG. If no optimum network can be found in LUT-based mapping, an alternative method needs to be used to synthesize the LUT. First, one can use the TO heuristics discussed in Section VI-D. Second, one can use majority-logic synthesis approaches, such as [10], that work on a truth table, and therefore, scale well for small functions.

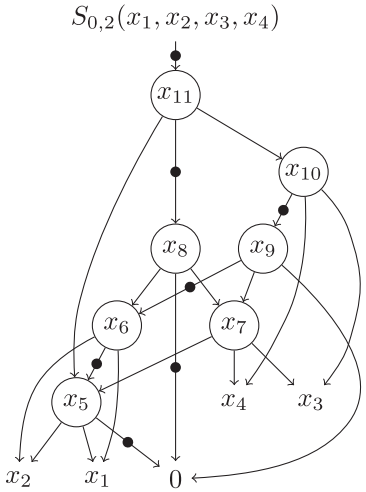


Fig. 7. Optimum MIG for $S_{0,2}(x_1, x_2, x_3, x_4)$.

C. Inverter Minimization

The inverter propagation property of the majority operation allows a large flexibility in arranging inverters in an MIG. For some technology dependent synthesis one is interested in MIGs with a small number of inverters, or some other characteristic.

For example, beyond-CMOS spintronic devices, such as spin transfer torque/domain wall devices [50], spin wave device [51], [52], or nanomagnetic logic [53] are majority-based. Consequently, MIGs can be used to derive spintronics realizations for given functions [12], [54]. Since the physical realization of an inverter is more expensive than the realization of the majority operation (from 1.5× to 4× depending on the technology [55]), one is interested in MIGs with a possibly small number of inverters.

The number of inverters can be constrained to a maximum value of P using the cardinality constraint

$$\sum_{n<i\leq n+r} p_{1i} + p_{2i} + p_{3i} \leq P. \tag{33}$$

Details on this encoding are provided in [56]. Notice that in this formulation we already make use of the optimization based on normal functions. This makes sense, since the existence of the output polarity solely depends on whether f is normal or not and cannot be adjusted using inverter propagation (when restricting the maximum number of inverted inputs for each majority gate to 1).

VIII. Experiments

We have implemented exact synthesis in C++ as command "*exact_mig*" in the logic synthesis framework CirKit using Z3 [57] as the underlying SMT solver.[2] The following sections provide details of the experimental evaluation. All experiments have been carried out on an Intel Xeon CPU E5-2680 v3 at 2.50 GHz with 64GB of main memory running Linux 4.4.

[2]The code can be downloaded from github.org/msoeken/cirkit, see also lsi.epfl.ch/MIG.

TABLE I
OPTIMUM MIGS FOR ALL 4-VARIABLE NPN CLASSES

			Size-optimum		Size/depth-optimum					Depth/size-optimum	
Gates	Classes	Functions	Time	Avg. time	Time	Avg. time	Depth	Classes	Functions	Time	Avg. time
0	2	10	0.00	0.00	0.00	0.00	0	2	10	0.00	0.00
1	2	80	0.02	0.01	0.04	0.02	1	2	80	0.02	0.01
2	5	640	0.13	0.03	0.24	0.05	2	48	10260	4.47	0.09
3	18	3300	1.10	0.06	2.15	0.12	3	169	55184	7745.58	2581.86
4	42	10352	5.65	0.13	11.16	0.27	4	1	2	>3h	
5	117	40064	64.93	0.01	126.67	1.08					
6	35	11058	108.23	3.09	187.67	5.36					
7	1	32	18.38	18.38	144.25	144.25					
Σ	222	65536	198.44		472.18			222	65536		

A. Computing Optimum MIGs

1) 4-Variable Functions: In order to compare exact synthesis for different cost objectives (i.e., size-optimum, size/depth-optimum, and depth/size-optimum), we computed all optimum MIGs for each objective and each representative of the 222 NPN classes using Algorithms 1–3. Table I lists the results.

The left part lists the results for size-optimum and size/depth-optimum synthesis partitioned by the number of majority gates. Two classes (the constant functions and the one-variable functions) do not require a majority gate. Another two classes (the two-variable AND and OR-like functions, as well as the MAJ-like functions) require one majority gate. All other functions require at least two majority gates. The representative of the single most difficult NPN class is the symmetric function $S_{0,2}(x_1, x_2, x_3, x_4) = \overline{x_1 \oplus x_2 \oplus x_3 \oplus x_4} \wedge \overline{x_1 x_2 x_3 x_4}$, which is NPN-equivalent to $S_{1,3,4} = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \vee x_1 x_2 x_3 x_4$ from Section VI-B. Its MIG representation is illustrated in Fig. 7. The overall runtime to obtain size-optimum MIGs for all 222 functions is a bit more than 3 min. Compared to the previous listed results [47] (18378.15 s), the use of symmetry breaking and other optimizations lead to a 92× speed-up. Additionally, considering depth-optimality as second criteria increases the total runtime by a factor of 2.4.

The right part of Table I lists the results for depth/size-optimum MIGs. The results are partitioned by the depth. Only for the NPN representation of the deepest NPN class (see [23], [47]) $S_{1,3}(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ we were not able to compute the optimum representation within 3 h. The problem are the unsatisfiable checks for HasMIG(f, r, 3) (see Algorithm 3) which have to be checked for $r \leq 13$ [see (22)]. This underlines the importance to find better upper bounds for the maximum number of possible gates at each level.

2) 6-Variable Functions: A similar enumerative experiment as in the previous section for 4-variable functions can be done for 5-variable functions in a very large but yet reasonable amount of time. There are 616 126 NPN classes that partition all 5-variable Boolean functions. Generously assuming that computing the optimum MIG for one class takes 10 min, one can find all optimum MIGs within 90 days when using 48 parallel processes. However, there are already over 200 trillion (200×10^{12}) NPN classes for 6-variable functions, which rules out enumeration.

We applied size-optimum exact synthesis using Algorithm 1 to a set of 6-variable functions. These functions have been mined from various benchmark sets by performing 6-cut enumeration on AIGs (see [58]). The functions have been partitioned based on whether they are disjoint-support decomposable (DSD, [59]). The function can be either fully decomposable (full-DSD), partly decomposable (part-DSD), or nondecomposable (non-DSD). The expectation is that fully decomposable functions are simpler, and therefore, easier for exact synthesis.

TABLE II
COMPUTATION TIME FOR 6-VARIABLE FUNCTIONS

Runtime (max.)	full DSD	part DSD	non DSD
10	87	466	500
20	29	521	1065
30	14	291	466
40	1	83	181
50	1	33	143
60	3	18	75
70	1	12	50
80	0	3	3
90	0	1	0
TO	65	822	3425
Σ	201	2250	5908
Success rate	67%	63%	42%

We run exact synthesis on each function, but give a TO of one minute to each call of HasMIG(f, r) in Algorithm 1. If the TO exceeds, the algorithm stops and reports a TO. Note that this is different from the TO used in Algorithm 4; there the algorithm does not terminate after the TO but increases the number of gates. Successful functions are partitioned based on their required runtime, where we take an upper limit that is a factor of 10. Note that the overall required runtime can exceed a minute, since the TO is given for an individual call of HasMIG(f, r).

The results are listed in Table II. For the full-DSD and part-DSD functions, most of the functions can be solved within the given time resources. For non-DSD functions, 42% of the instances led to an optimum MIG before the TO. For all three classes of functions, a common trend is observable. If an optimum MIG can be found, then for most of the successful instances it can be found within less than 30 s. This motivates to use exact synthesis as a first attempt and resort to heuristics if not successful after a short time.

B. Application to MIG Size Optimization

We used functional hashing (Section VII-A) and LUT-based optimization (Section VII-B) to reduce the size instances of the EPFL combinational benchmark suite.[3] Both approaches make use of size-optimum exact synthesis by replacing subnetworks with their optimum counterparts. Subnetworks with up to four

[3] lsi.epfl.ch/benchmarks

TABLE III
USING EXACT SYNTHESIS FOR MIG-SIZE OPTIMIZATION

		Initial size		**Func. hash. [47]**		**LUT-based size opt. [49]**	
Benchmark	I/O	Size	Depth	Size	Depth	Size	Depth
Adder	256/129	1020	255	1020	255	513	130
Barrel shifter	135/128	3336	12	3240	15	3336	12
Divisor	128/128	57247	4372	57247	4403	35993	3607
Hypotenuse	256/128	214335	24801	214307	24878	196842	11317
Log2	32/32	32060	444	29795	461	32060	444
Max	512/130	2865	287	2865	337	2479	276
Multiplier	128/128	27062	274	24903	271	22634	165
Sine	24/25	5416	225	5254	230	5416	255
Square-root	128/64	24618	5058	24618	6021	24170	6073
Square	64/128	18484	250	17893	250	17562	130
Average improvement (new/old):				0.97	1.07	**0.87**	**0.81**
Round-robin arbiter	256/129	11839	87	11839	88	8957	63
Alu control unit	7/26	174	10	149	10	139	10
Coding-cavlc	10/11	693	16	693	16	757	19
Decoder	8/256	304	3	304	3	328	4
i2c controller	147/142	1342	20	1342	21	1329	23
Int to float converter	11/7	260	16	260	17	263	18
Memory controller	1204/1231	46836	114	46568	137	45034	144
Priority encoder	128/8	978	250	976	249	993	245
Lookahead XY router	60/30	257	54	248	54	220	54
Voter	1001/1	13758	70	13587	74	7767	67
Average improvement (new/old):				0.98	1.04	0.91	1.07

TABLE IV
USING EXACT SYNTHESIS FOR LUT OPTIMIZATION COMPARED TO BEST KNOWN PREVIOUS RESULTS

		Best area		**LUT-based size opt. [49]**		**Best depth**		**Func. hash. [47]**	
Benchmark	I/O	Area	Delay	Area	Delay	Area	Delay	Area	Delay
Adder	256/129	201	73	**192**	64	419	6	418	**6**
Barrel shifter	135/128	512	4	512	4	521	4	n/a	n/a
Divisor	128/128	3813	1542	10328	1915	14576	238	14172	**238**
Log2	32/32	7344	142	7592	131	9275	55	9162	**54**
Max	512/130	532	192	692	113	899	10	888	**10**
Multiplier	128/128	5681	120	**5192**	102	7095	29	7051	**29**
Sine	24/25	1347	62	1402	66	1835	30	1801	**30**
Square-root	128/64	3286	1180	6810	2070	11745	254	11758	255
Square	64/128	3800	116	**3309**	74	4203	11	4154	**11**

variables are considered for which all optimum networks are precomputed.

Table III shows the results. The average size improvements are 3% and 13% for the arithmetic instances with functional hashing and LUT-based optimization, respectively. LUT-based optimization uses LUTs to decompose the circuit while functional hashing uses cuts within fanout-free regions. The experiments suggest that the decomposition into LUTs is more advantages for exact synthesis. This is especially evident when comparing the depth, which is increased when using functional hashing but decreased significantly with LUT-based optimization.

C. Application to LUT Mapping

Next, we show the effectiveness of the exact synthesis-based optimization approaches to 6-LUT mapping. The EPFL benchmarks come with the currently best known realizations (version 2015.1), both with respect to area and delay. Note that the reported numbers have been obtained from a variety of different techniques and therefore the aim is not to necessarily improve on all benchmarks, but to advance the state-of-the-art by improving on some of them.

We used LUT-based size optimization [49] using size-optimum exact synthesis (for subfunctions with up to four variables) as engine to decrease the area in the designs that show best area results. The size optimized MIGs are read with ABC and mapped using "*if −K 6 −a*." As shown by Table IV, area can be improved for three out of the considered nine benchmarks.

We compared the best results reported in [47] obtained using functional hashing to the best delay results in the EPFL benchmarks (for the barrel shifter no results were reported). Functional hashing is able to reduce delay for one of the benchmarks (Log2). However, in six further cases, one can observe a reduction in area while depth is kept the same. We also account this for an improvement over the state-of-the art.

IX. CONCLUSION

We have presented an algorithm for exact synthesis of MIGs based on SMT solving. The advantage of the encoding is its flexibility to handle different cost criteria and be extensible to different scenarios, e.g., inverter minimization. Various optimization techniques including symmetry breaking led to a significant improvement in runtime and extend the applicability to functions up to six variables, a size for which explicit enumeration is infeasible. We described several applications in which exact synthesis is embedded into a broader scope and can therefore be applied to larger functions.

With exact synthesis and its applications, we are able to improve best known results of the arithmetic benchmarks in the EPFL suite for area in three out of nine cases and for delay in seven out of eight cases. The current best results are produced by the strongest AIG and MIG optimization scripts from Berkeley and EPFL groups. Consequently, our findings and new results advance the state-of-the-art in logic optimization.

ACKNOWLEDGMENT

The authors wish to thank A. Mishchenko, W. Haaswijk, and E. Testa for their many helpful discussions.

REFERENCES

[1] V. Kabanets and J.-Y. Cai, "Circuit minimization problem," in *Proc. 32nd Annu. ACM Symp. Theory Comput.*, Portland, OR, USA, 2000, pp. 73–79.

[2] C. D. Murray and R. R. Williams, "On the (non) NP-hardness of computing circuit complexity," in *Proc. 30th Conf. Comput. Complexity*, Portland, OR, USA, 2015, pp. 365–380.

[3] A. A. Razborov and S. Rudich, "Natural proofs," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 24–35, 1997.

[4] E. A. Ernst, "Optimal combinational multi-level logic synthesis," Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. Michigan, Ann Arbor, MI, USA, 2009.

[5] S. Muroga and T. Ibaraki, "Design of optimal switching networks by integer programming," *IEEE Trans. Comput.*, vol. 21, no. 6, pp. 573–582, Jun. 1972.

[6] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Boston, MA, USA: Addison-Wesley, 2015.

[7] L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Proc. 51st Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2014, pp. 1–6.

[8] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1377–1394, Dec. 2002.

[9] L. Hellerman, "A catalog of three-variable or-invert and-invert logical circuits," *IEEE Trans. Electron. Comput.*, vol. 12, no. 3, pp. 198–223, Jun. 1963.

[10] S. B. Akers, Jr., "Synthesis of combinational logic using three-input majority gates," in *Proc. 3rd Annu. Symp. Switching Circuit Theory Logical Design (SWCT)*, Chicago, IL, USA, 1962, pp. 149–157.

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

[12] R. Zhang, P. Gupta, and N. K. Jha, "Majority and minority network synthesis with application to QCA-, SET-, and TPL-Based nanotechnologies," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1233–1245, Jul. 2007.

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

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

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

[16] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "DPLL(T): Fast decision procedures," in *Proc. 16th Int. Conf. CAV*, Boston, MA, USA, 2004, pp. 175–188.

[17] R. M. Karp, F. E. McFarlin, J. P. Roth, and J. R. Wilts, "A computer program for the synthesis of combinational switching circuits," in *Proc. 2nd Annu. Symp. Switching Circuit Theory Logical Design (SWCT)*, Detroit, MI, USA, 1961, pp. 182–194.

[18] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM J. Res. Develop.*, vol. 6, no. 2, pp. 227–238, Apr. 1962.

[19] P. R. Schneider and D. L. Dietmeyer, "An algorithm for synthesis of multiple-output combinational logic," *IEEE Trans. Comput.*, vol. 17, no. 2, pp. 117–128, Feb. 1968.

[20] E. L. Lawler, "An approach to multilevel Boolean minimization," *J. ACM*, vol. 11, no. 3, pp. 283–295, 1964.

[21] R. A. Smith, "Minimal three-variable NOR and NAND logic circuits," *IEEE Trans. Electron. Comput.*, vol. 14, no. 1, pp. 79–81, Feb. 1965.

[22] R. Drechsler and W. Günther, "Exact circuit synthesis," in *Proc. Int. Workshop Logic Synth.*, 1998.

[23] D. E. Knuth, *The Art of Computer Programming, Volume 4A*. Reading, MA, USA: Addison-Wesley, 2011.

[24] C. R. Baugh, T. Ibaraki, and S. Muroga, "Technical note—Results in using Gomory's all-integer integer algorithm to design optimum logic networks," *Oper. Res.*, vol. 19, no. 4, pp. 1090–1096, 1971.

[25] C. R. Baugh, C. S. Chandersekaran, R. S. Swee, and S. Muroga, "Optimal networks of NOR-OR gates for functions of three variables," *IEEE Trans. Comput.*, vol. 21, no. 2, pp. 153–160, Feb. 1972.

[26] S. Muroga and H. C. Lai, "Minimization of logic networks under a generalized cost function," *IEEE Trans. Comput.*, vol. 25, no. 9, pp. 893–907, Sep. 1976.

[27] A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, "Finding efficient circuits using SAT-solvers," in *Proc. 12th Int. Conf. SAT*, Swansea, U.K., 2009, pp. 32–44.

[28] N. Éen, "Practical SAT—A tutorial on applied satisfiability solving," in *Proc. FMCAD*, Austin, TX, USA, 2007.

[29] E. S. Davidson, "An algorithm for NAND decomposition under network constraints," *IEEE Trans. Comput.*, vol. 18, no. 12, pp. 1098–1109, Dec. 1969.

[30] J. N. Culliney, M. H. Young, T. Nakagawa, and S. Muroga, "Results of the synthesis of optimal networks of AND and OR gates for four-variable switching functions," *IEEE Trans. Comput.*, vol. 28, no. 1, pp. 76–85, Jan. 1979.

[31] L. G. Amarù, P.-E. Gaillardon, A. Chattopadhyay, and G. De Micheli, "A sound and complete axiomatization of majority-n logic," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2889–2895, Sep. 2016.

[32] A. Chattopadhyay, L. G. Amarù, M. Soeken, P.-E. Gaillardon, and G. De Micheli, "Notes on majority Boolean algebra," in *Proc. IEEE 46th Int. Symp. Multiple Valued Logic (ISMVL)*, Sapporo, Japan, 2016, pp. 50–55.

[33] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 205–213, Feb. 1991.

[34] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," Dept. Elect. Eng. Comput. Sci., Univ. California at Berkeley, Berkeley, CA, USA, Tech. Rep., 2005.

[35] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 8, pp. 813–834, Aug. 1997.

[36] S. Chatterjee, A. Mishchenko, R. K. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2894–2903, Dec. 2006.

[37] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for LUT-based FPGAs," in *Proc. ACM/SIGDA 6th Int. Symp. Field Program. Gate Arrays*, Monterey, CA, USA, 1998, pp. 35–42.

[38] A. Mishchenko, S. Cho, S. Chatterjee, and R. K. Brayton, "Combinational and sequential mapping with priority cuts," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2007, pp. 354–361.

[39] S. Muroga, *Logic Design and Switching Theory*. New York, NY, USA: Wiley, 1979.

[40] L. Benini and G. De Micheli, "A survey of Boolean matching techniques for library binding," *ACM Trans. Design Autom. Electron. Syst.*, vol. 2, no. 3, pp. 193–226, 1997.

[41] R. Nieuwenhuis and A. Oliveras, "On SAT modulo theories and optimization problems," in *Proc. 9th Int. Conf.*, Seattle, WA, USA, 2006, pp. 156–169.

[42] G. S. Tseytin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics*, A. P. Slisenko, Ed. New York, NY, USA: Springer, 1970, pp. 115–125.

[43] N. Eén, A. Mishchenko, and N. Sörensson, "Applying logic synthesis for speeding up SAT," in *Proc. 10th Int. Conf.*, Lisbon, Portugal, 2007, pp. 272–286.

[44] O. B. Lupanov, "A method of circuit sythesis," *Isvestiia VUZov Radiofizika*, vol. 1, pp. 120–140, 1958.

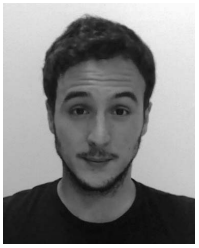
[45] R. K. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. 22nd Int. Conf. CAV*, Edinburgh, U.K., 2010, pp. 24–40.

[46] N. Lodha, S. Ordyniak, and S. Szeider, "A SAT approach to branchwidth," in *Proc. 19th Int. Conf.*, Bordeaux, France, 2016, pp. 179–195.
[47] M. Soeken, L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Optimizing majority-inverter graphs with functional hashing," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Dresden, Germany, 2016, pp. 1030–1035.
[48] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 1, pp. 1–12, Jan. 1994.
[49] W. Haaswijk, M. Soeken, L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic optimization," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2017.
[50] J. A. Currivan, Y. Jang, M. D. Mascaro, M. A. Baldo, and C. A. Ross, "Low energy magnetic domain wall logic in short, narrow, ferromagnetic wires," *IEEE Magn. Lett.*, vol. 3, 2012, Art. no. 3000104.
[51] A. Khitun and K. L. Wang, "Nano scale computational architectures with spin wave bus," *Superlattices Microstruct.*, vol. 38, no. 3, pp. 184–200, 2005.
[52] A. Khitun *et al.*, "Inductively coupled circuits with spin wave bus for information processing," *J. Nanoelectron. Optoelectron.*, vol. 3, no. 1, pp. 24–34, 2008.
[53] R. P. Cowburn and M. E. Welland, "Room temperature magnetic quantum cellular automata," *Science*, vol. 287, no. 5457, pp. 1466–1468, 2000.
[54] Z. Huo, Q. Zhang, S. Haruehanroengra, and W. Wang, "Logic optimization for majority gate-based nanoelectronic circuits," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2006, pp. 1307–1310.
[55] D. E. Nikonov and I. A. Young, "Benchmarking of beyond-CMOS exploratory devices for logic integrated circuits," *IEEE J. Exploratory Solid-State Comput. Devices Circuits*, vol. 1, pp. 3–11, 2015.
[56] E. Testa *et al.*, "Inversion optimization in majority-inverter graphs," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit. (NANOARCH)*, Beijing, China, 2016, pp. 15–20.
[57] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. TACAS/ETAPS*, Budapest, Hungary, 2008, pp. 337–340.
[58] Z. Huang, L. Wang, Y. Nasikovskiy, and A. Mishchenko, "Fast Boolean matching based on NPN classification," in *Proc. Int. Conf. Field Program. Technol. (FPT)*, Kyoto, Japan, 2013, pp. 310–313.
[59] V. Bertacco and M. Damiani, "Boolean function representation based on disjoint-support decompositions," in *Proc. IEEE Int. Conf. Comput. Design VLSI Comput. Processors (ICCD)*, Austin, TX, USA, 1996, pp. 27–32.

Mathias Soeken (S'09–M'13) received the Ph.D. degree in computer science and engineering from the University of Bremen, Bremen, Germany, in 2013.

He is a Scientist with the École Polytechnique Fédéderale de Lausanne, Lausanne, Switzerland. He is investigating constraint-based techniques in logic synthesis and industrial-strength design automation for quantum computing. He is involved in active collaborations with University of California at Berkeley, Berkeley, CA, USA, and Microsoft Research, Redmond, WA, USA. He is also actively maintaining the logic synthesis frameworks CirKit and RevKit. His current research interests include logic synthesis and formal verification.

Dr. Soeken was a recipient of the Scholarship from the German Academic Scholarship Foundation. He has been serving as a TPC Member for several conferences, including DAC'17 and ICCAD'17 and is a Reviewer for Mathematical Reviews as well as for several journals.



Luca Gaetano Amarù (S'13–M'16) received the B.S. and M.S. degrees in electronic engineering from the Politecnico di Torino, Turin, Italy, in 2009 and 2011, respectively, and the Ph.D. degree in computer science from the Swiss Federal Institute of Technology Lausanne, Lausanne, Switzerland, in 2015.

He is a Senior II Research and Development Engineer with the Design Group of Synopsys Inc., Mountain View, CA, USA, where he is responsible for designing efficient data structures and algorithms for logic synthesis. His current research interests include electronic design automation, logic in computer science, and beyond CMOS technologies.

Dr. Amaru was a recipient of the EDAA Outstanding Dissertation Award, in 2015, the Best Presentation Award at FETCH Conference, in 2013, and the Best Paper Award Nomination at ASP-DAC conference, in 2013.

Pierre-Emmanuel Gaillardon (S'10–M'11–SM'16) received the Electrical Engineer degree from CPE-Lyon, Villeurbanne, France, in 2008, the M.Sc. degree in electrical engineering from INSA Lyon, Villeurbanne, in 2008, and the Ph.D. degree in electrical engineering from the University of Lyon, Lyon, France, in 2011.

He was a Research Associate with the École Polytechnique Fédéderale de Lausanne, Lausanne, Switzerland, and a Research Assistant with CEA-LETI, Grenoble, France. He is an Assistant Professor with the ECE Department, University of Utah, Salt Lake City, UT, USA. His current research interests include the development of reconfigurable logic architectures and digital circuits exploiting emerging device technologies and novel EDA techniques.

Prof. Gaillardon is an Associate Editor of the IEEE TRANSACTIONS ON NANOTECHNOLOGY. He has been serving as a TPC Member for many conferences, including DATE'15-17, DAC'17, and is a Reviewer for several journals. He serves as the Topic Co-Chair for DATE'17.

Giovanni De Micheli (M'83–SM'89–F'94) received the Nuclear Engineering degree from the Politecnico di Milano, Milan, Italy, in 1979 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California at Berkeley, Berkeley, CA, USA, in 1980 and 1983, respectively.

He was a Professor of Electrical Engineering with Stanford University, Stanford, CA, USA. He is a Professor and the Director of the Institute of Electrical Engineering, École Polytechnique Fédéderale de Lausanne, Lausanne, Switzerland.

Mr. De Micheli is a recipient of the 2016 IEEE/CS Harry Goode Award for seminal contributions to design and design tools of Networks on Chips, the 2016 EDAA Lifetime Achievement Award, and other awards. He is a fellow of ACM and a member of the Academia Europaea.