Structural Rewriting in XOR-Majority Graphs

Zhufei Chu¹ Mathias Soeken² Yinshui Xia¹ Lunyao Wang¹ Giovanni De Micheli²

¹Faculty of Electrical Engineering & Computer Science, Ningbo University, Ningbo, China
²École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

ABSTRACT
In this paper, we present a structural rewriting method for a recently proposed XOR-Majority graph (XMG), which has exclusive-OR (XOR), majority-of-three (MAJ), and inverters as primitives. XMGs are an extension of Majority-Inverter Graphs (MIGs). Previous work presented an axiomatic system, Ω, and its derived transformation rules for manipulation of MIGs. By additionally introducing XOR primitive, the identities of MAJ-XOR operations should be exploited to enable powerful logic rewriting in XMGs. We first proposed two MAJ-XOR identities and exploit its potential optimization opportunities during structural rewriting. Then, we discuss the rewriting rules that can be used for different operations. Finally, we also address structural XOR detection problem in MIG. The experimental results on EPFL benchmark suites show that the proposed method can optimize the size/depth product of XMGs and its mapped look-up tables (LUTs), which in turn benefits the quantum circuit synthesis that using XMG as the underlying logic representations.

CCS CONCEPTS
• Hardware → Combinational synthesis; Circuit optimization;

KEYWORDS
logic synthesis, logic networks, rewriting, majority logic

1 INTRODUCTION
Multi-level logic synthesis plays an important role in automated design flow [7, 13]. It aims at finding a multi-level logic network of Boolean function in terms of better quality while considering different cost functions. Typical cost functions are the number of logic gates, logic depth, and switching activity, which in turn corresponded to better area, performance, and energy. To this end, efficient representation and optimization of Boolean functions are key features [1].

With the continuous increase in logic design complexity, the logic representations have shifted from complex heterogenous one to simpler homogenous networks, such as And-Inverter Graphs (AIGs) [14] and Majority-Inverter Graphs (MIGs) [1, 3]. These homogenous representations are simple and easy to manipulate, which enable more efficient optimization, requiring less memory and allowing better run times [10]. Logic rewriting is carried out to optimize logic representations, which is the most scalable optimization strategy and can be applied to very large functions. In terms of MIG rewriting, it can be either functional, e.g., by replacing small subnetworks up to four inputs with their optimum counterpart exploiting the results from exact synthesis [15], or structural, e.g., by applying the five transformation rules in an axiomatic system, Ω, and its derived three transformation rules, Ψ [1].

To obtain a more compact logic representations, it is reported that the introduction of exclusive-OR (XOR) operation can be significantly more advantageous over homogenous logic representations. Therefore, the extensions of AIG and MIG to XOR-AIG (XAIG) [11] and XOR-MIG (XMG) [10] are presented, respectively. XMGs have been applied in exact synthesis aware rewriting, pre-optimization for 6-LUT mapping [10], and synthesis of quantum networks. Especially, in commonly used cost models for quantum computing, majority-of-three (MAJ) can be implemented as the same cost of AND/OR and the cost of an XOR can be neglected [16]. Hence, XMGs are advantageous for quantum circuit synthesis. In order to support the natural manipulation of MIGs, a new Boolean algebra consisting of Ω and Ψ is proposed based exclusively on MAJ and inverter operations. However, a thorough consideration of MAJ-XOR logic expressions and their implementation in XMGs are not addressed in the literature.

The current XMG generation method in the literature is based on functional rewriting (FR) of AIGs [10], which is further improved by functional decomposition using MAJ [9]. Given an input network N, the approach proposed in [10] first maps the network into k-LUTs, e.g., in a size- or depth-oriented manner. Each k-LUT represents a k-variable Boolean function which is then used as input for exact synthesis. The results of exact synthesis are saved. Then the locally optimum networks are merged together to construct an optimized, functionally equivalent network N'.

In this paper, we aim to exploit structural rewriting (SR) method based on Ω and several MAJ-XOR identities that favor optimizations of size, depth, or inverters configuration in XMGs. Our contributions are as follows:

(1) We propose two MAJ-XOR identities and exploit its potential optimization opportunities during SR. Also, we discuss the rewriting rules that can be used for different operations in XMG. (Section 3).
(2) In contrast with FR method, we propose XOR structural detection method in MIG to build an XMG. (Section 4)
(3) Based on MAJ-XOR identities and XOR structural detection, we present an XMG size optimization algorithm. (Section 5)
Given an XMG obtained by FR as a starting point, experimental results on EPFL benchmarks reveal that the proposed SR method achieves with an average 32% reduction on XMG size/depth product, while 5% reduction on look-up tables (LUT) size/depth product. Also, considering the implementation cost of a T gate is extremely expensive in quantum circuit realization, the results on quantum synthesis show that the proposed method can optimize T gate count by 6% while using 5% more quantum bits (also called qubits or lines).

2 MAJORITY-INVERTER GRAPHS

Majority is a powerful generalization of AND/ORs, the MAJ function evaluates to true if and only if at least two variables are true. It can be expressed in disjunctive, conjunctive normal form, and exclusive-sum-of-products (ESOP) form as

\[
\Phi(x,y,z) = x \lor y \lor z = (x \lor y) \lor z
\]

where \(\land\) is the XOR operation as

\[
x \land y = x \lor y = (x \lor y)(\bar{x} \lor \bar{y})
\]

By setting one of its arguments to 0 or 1, the Boolean operation AND and OR can be obtained from MAJ, respectively.

\[
(0xy) = x \land y \quad \text{and} \quad (1xy) = x \lor y
\]

As a homogenous logic representation, MIGs use MAJ together with negation (inverter) as the only logic operations. Hence, the general AND/OR/Inverter Graphs (AOIGs) or AIGs are a special case of MIGs. MIG-based representations are extremely effective at logic rewriting. The axiomatic system for the MIG Boolean algebra, referred to as \(\Omega\) (Eqn. (4)), is defined by five primitive transformation rules: commutativity (\(\Omega.C\)), majority (\(\Omega.M\)), associativity (\(\Omega.A\)), distributivity (\(\Omega.D\)), and inverter propagation (\(\Omega.I\)).

\[
\Omega = \begin{cases}
\text{Commutativity} & - \Omega.C \\
(xy) = (yx) = (zx)
\end{cases}
\]

\[
\begin{cases}
\text{Majority} & - \Omega.M \\
(xy) = \bar{z} & \text{if } x = y \\
(xy) = z & \text{if } x = \bar{y}
\end{cases}
\]

\[
\begin{cases}
\text{Associativity} & - \Omega.A \\
(xy(uz)) = ((xy)uz)
\end{cases}
\]

\[
\begin{cases}
\text{Distributivity} & - \Omega.D \\
(xy(uz)) = ((xy)uz)
\end{cases}
\]

\[
\begin{cases}
\text{Inverter Propagation} & - \Omega.I \\
(xy) = (x\bar{y})
\end{cases}
\]

A strong property of MIGs and their algebraic framework is reachability. It has been proven that, by using a sequence of transformations drawn from the primitive five rules, it is possible to traverse the entire MIG representation space [1]. Rewriting strategies have been developed based on these rules which allow size reduction and significant depth reduction. The rewriting algorithm can be implemented more efficiently and more effectively when taking derived transformation rules into consideration. It turns out that the following three, referred to as \(\Psi\) (Eqn. (5)), are particularly helpful.

\[
\Psi = \begin{cases}
\text{Relevance} & - \Psi.R \\
(xy) = (x\bar{y})
\end{cases}
\]

\[
\begin{cases}
\text{Complementary Associativity} & - \Psi.C \\
(xy(uz)) = ((x\bar{y})uz)
\end{cases}
\]

\[
\begin{cases}
\text{Substitution} & - \Psi.S \\
(xy\bar{z}) = ((x\bar{y})\bar{z})(\bar{x}y\bar{z})
\end{cases}
\]

where \(f_{x\bar{y}}\) is the expression that is obtained when replacing all occurrences of \(x\) with \(y\) in \(f\).

3 XOR-MAJORITY GRAPHS

In this section, we first present XMGs and then exploit their associated Boolean algebra. Notable properties of XMGs are discussed.

3.1 XMG Logic Representation

The main motivation to extend MIG into XMG is the complexity of expressing the \(n\)-input parity function using MIG. The optimal MIG-based representation of \(4\)-input parity function \(f = x_1 \oplus x_2 \oplus x_3 \oplus x_4\), obtained by exact synthesis, requires nine MAJ operations as

\[
f = \langle x_1 \langle x_2 \langle x_3 \langle x_2 \bar{x}_2 x_4 \rangle \bar{x}_2 \rangle \rangle \rangle
\]

The complexity grows quickly when we add even more inputs. Efficient logic realization of arithmetic components heavily depend on 3-input parity functions. Hence, XMG that contain both MAJ and XOR can be significantly more advantageous over MIG, as shown in Fig. 1, in which each node corresponds to either MAJ (3-input) or XOR (2-input) operator and the connections between nodes can be inverted.

3.2 XMG Boolean Algebra

XOR operation has several useful properties. The basic identities are as

\[
\Delta = \begin{cases}
x \oplus y = 0 & \text{if } x = y \\
x \oplus y = 1 & \text{if } x = \bar{y} \\
x \oplus y = 0 & x = x \\
x \oplus y = 1 & x = \bar{x}
\end{cases}
\]

We refer to XOR Boolean algebra as \(\Phi\). XOR operations are associative (\(\Phi.A\)) and commutative (\(\Phi.C\)). Although it is not self-dual, it also allows to propagate inverters (\(\Phi.I\)).
With this simplification, we conclude the proof.

\[ \Phi \]

\[ \Phi \]

\[ \Phi \]

\[ \Phi \]

Further, the XOR does not distribute over any other binary operation, but logic conjunction (and) does distribute over XOR.

\[ x(y \oplus z) = xy \oplus xz \]

Combined MAJ and XOR operations, denoted as MAJ-XOR, next we exploit the Boolean algebra of MAJ-XOR operations by providing a list of identities.

**Theorem 1.** XOR operation distribute over MAJ function.

\[ \langle xyz \rangle \oplus u = \langle (x \oplus u)(y \oplus u)(z \oplus u) \rangle \]

**Proof.** We expand the right hand side (RHS) expression by majority function definition in ESOP form, that is

\[ \langle (x \oplus u)(y \oplus u) \rangle \]

\[ \oplus \langle (x \oplus u)(z \oplus u) \rangle \oplus \langle (y \oplus u)(z \oplus u) \rangle \]

\[ Eqn.(9) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(7) \]

With this simplification, we conclude the proof.

By applying Eqn. (10) from RHS to LHS (left hand side), the size (node number used in XMG) is reduced from 4 to 2, while the shared input is pushed up one level, as shown in Fig. 2. Consequently, it is possible to rewrite a MAJ node for size and depth optimization, if (i) all children are the same nodes type XOR, and (ii) there is one shared input to these XOR nodes.

The constraints to apply Eqn. (10) is tight, but there are several special cases in Eqn. (10), which may be helpful as a relaxation of constraints. We list the identities in the following item, due to the symmetry, we only list the special cases of the variable \( x \).

(1) The shared input \( u \) equals to one of the other three inputs \( x, y, z \). Since the XOR operation over the same variable would result in constants, it provides a special way to deal with MAJ nodes with constant inputs.

\[ u = x \Rightarrow (0(y \oplus u)(z \oplus u)) = u \oplus (uyz) \]

\[ u = x \Rightarrow (1(y \oplus u)(z \oplus u)) = u \oplus (uyz) \]

\[ Eqn.(11) \]

(2) One of the other three inputs \( x, y, z \) equals constant 0 or 1. That provides a way to deal with MAJ nodes with two XOR-type children and one direct shared input in arbitrary polarity.

\[ x = 0 \Rightarrow (u(y \oplus u)(z \oplus u)) = u \oplus (0yz) \]

\[ x = 1 \Rightarrow (u(y \oplus u)(z \oplus u)) = u \oplus (1yz) \]

\[ Eqn.(12) \]

(3) Both above two cases happened, suppose \( u \) equals \( x \) with arbitrary polarity and \( y \) is constant, we obtained

\[ u = x, y = 0 \Rightarrow (0u(z \oplus u)) = u \oplus (0uz) \]

\[ u = x, y = 1 \Rightarrow (1u(z \oplus u)) = u \oplus (1uz) \]

\[ u = x, y = 0 \Rightarrow (1u(z \oplus u)) = u \oplus (1uz) \]

**Theorem 2.** Complementary associativity rule exists in MAJ-XOR operations.

\[ \langle xy(\bar{y} \oplus z) \rangle = \langle xy(x \oplus z) \rangle \]

**Proof.** We expand the LHS and RHS of Eqn. (14) in ESOP form, respectively, that is

\[ LHS = \langle (x(\bar{y} \oplus z)) \oplus (y(\bar{y} \oplus z)) \rangle \oplus xy \]

\[ Eqn.(9) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(7) \]

\[ Eqns.(9),(7) \]

\[ Eqns.(9),(7) \]

\[ Eqns.(9),(7) \]

\[ Eqns.(9),(7) \]

\[ Eqns.(9),(7) \]

\[ Eqns.(9),(7) \]

\[ Eqns.(9),(7) \]

RHS = \langle (x(\bar{y} \oplus z)) \oplus (y(x \oplus z)) \rangle \oplus xy

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn.(7) \]

\[ Eqn.(9) \]

\[ Eqn(14) \]

Therefore, LHS = RHS, which concludes the proof.

**Theorem 2** is used to deal with reconvergent variables appearing both polarities. As shown in Fig. 3, the reconvergent variable \( y \) is pushed up one level, and the inverter is also removed. Therefore, it provides optimization opportunities in depth and number of inverters. In terms of size, if we consider special case that variable \( x \) is constant 0 and 1, we can obtain the already know identities, that are

\[ x = 0 \Rightarrow y(\bar{y} \oplus z) = yz \]

\[ x = 1 \Rightarrow y \vee (\bar{y} \oplus z) = y \vee z \]

\[ Eqn.(15) \]
To specify one of variables from Eqn. (16) as constant input 0 or 1, we obtain following identities.

As constant input. For example, we replace one of variables \{polarities, but as a combination of one variable with the other one shown in Fig. 4, there are several algorithms to escape local minima. The XOR detection method can be adopted during the optimization procedure can be iterated over a user-defined number of cycles, called effort. The XOR detection method can be adopted during the loop, which is like permutation of solutions used in evolutionary algorithms to escape local minima. In terms of XMG size optimization, we can also use \(\Omega.M_{L\rightarrow R}\) and \(\Omega.D_{R\rightarrow L}\) for MAJ-XOR nodes elimination. For XOR nodes, from the identities presented in \(\Delta\), if the two inputs of XOR are the same with arbitrary polarities, then the number of XOR nodes can be reduced. The above strategies are used separately for MAJ and XOR nodes, respectively. With the target of MAJ-XOR nodes, the node elimination opportunity arises from the identity shown in Eqn. (10), evaluated from right to left.

To reshape the XMG, as we discussed in Section III.3.3, only \(\Omega.A\) and \(\Psi.C\) can be applied for MAJ nodes. Besides, the push-up axioms proposed in [3] are also used for MIG depth optimization. The \(\Phi.A\) can be applied to XOR nodes, and the identity shown in Eqn. (14) can be used for MAJ-XOR nodes. The elimination and reshaping procedure can be iterated over a user-defined number of cycles, called effort. The XOR detection method can be adopted during the iteration or implemented at the beginning but just once. However, the experimental results show the former case may consume too much CPU time. In contrast, the latter case is more reasonable. If the experimental results show the former case may consume too much CPU time. In contrast, the latter case is more reasonable.

To reshape the XMG, as we discussed in Section III.3.3, only \(\Omega.A\) and \(\Psi.C\) can be applied for MAJ nodes. Besides, the push-up axioms proposed in [3] are also used for MIG depth optimization. The \(\Phi.A\) can be applied to XOR nodes, and the identity shown in Eqn. (14) can be used for MAJ-XOR nodes. The elimination and reshaping procedure can be iterated over a user-defined number of cycles, called effort. The XOR detection method can be adopted during the iteration or implemented at the beginning but just once. However, the experimental results show the former case may consume too much CPU time. In contrast, the latter case is more reasonable.
We implemented our approach in C++ as a command called ‘xmgre’ on top of the logic synthesis framework CirKit.1 The benchmarks considered are general combinational circuits from ISCAS [8] and EPFL benchmark suites [2]. Our SR results are verified by ‘cecc’ command in ABC [6] to ensure functional correctness.

### 6.2 Results

**Evaluation on EPFL Benchmarks.** The experimental results are shown in Table 1. The “Benchmarks” column lists the benchmark name. Given the results produced by ‘xmglut -k 4’ in CirKit [10], our SR algorithm is applied to further optimize XMG, which is a post optimization strategy. The EPFL benchmark suites contain 20 benchmarks. We list 17 of them while the remaining 3 got exactly 0% for the SR result.

In terms of XMG size, the number of nodes can be reduced by 3% on average, while the XMG depth is reduced by 22%, compared to [10]. The improvement of the results are highlighted in the table. Among 17 benchmarks, 16 of them can be optimized in size and 6 in depth. Generally, the XMG size can be reduced with an overhead of XMG depth. The exceptions are benchmarks div, sqrt, arbiter, ctrl, and priority, which achieves both size and depth improvement. For instance, arbiter achieves up to 12.65% size improvement, from 12,167,283 to 1,276,777 nodes, and 14.61% depth improvement, from 12,232 to 8,46 nodes, which is equal 41.27% reduction. The latter one also got a 14.61% depth reduction.

We set the rewriting algorithm terminated when the nodes cannot be improved after at least two efforts. Therefore, the rewriting cycles are distinct for benchmarks. The CPU time listed in the table indicates our SR method averagely consumes 2.29 seconds.

We also compares the results after 6-LUT mapping. Generally, XMG size optimization advantage also carries over into LUT mapping improvements in a vast majority cases. However, optimization of the size and depth of a logic network may not essentially result in reduced LUT size and depth [12]. For example, benchmarks max and mem_ctrl can be optimized in terms of XMG size, whereas results in an increase of LUT size. On average, our method achieves 3% reduction of LUT size and 1% reduction of LUT depth.

By evaluating the size/depth product metric, our method achieves 32% reduction of XMG, while 5% reduction of LUT after technology mapping. The CPU time listed in the table indicates our SR method averagely consumes 2.29 seconds.

### 6.3 Results

**Evaluation on ISCAS Benchmarks.** Since XOR detection method is proposed, we can also proceed SR directly on AIGs, which can be transposed into MIGs by adding constant inputs. To compare the proposed rewriting method for size optimization, we compare several rewriting method using the same ISCAS benchmarks. The results are shown in Fig. 5, where the MIG rewriting method are used as baseline. The detail commands used in Cirkit to implement these methods are shown below, while the start point is reading a circuit file in AIGER format [4].

- `mig rewrite --metric 1`
- `xmgre
- `xmglut -k 4`
Although the SR method can optimize XMG size, due to the de-
pend on the technology mapping results. Through efficient XOR
detection, we can even achieve better results than FR method.
For example, the XMG size of benchmark c1908 is 227 by FR, while 226
by SR.

Despite these circuits where FR performed less well than the
original ones, FR method generally produce better results than
our SR method. The main reason is technology mapping used FR
already adopts intelligence of area- and depth-oriented optimization.
Therefore, it makes sense to exploit “FR + SR” simultaneously to
guarantee eventually better results.

6.3 Evaluation on Quantum Circuit Synthesis
XMGs have been applied in quantum circuit synthesis [16]. The
basic principle is to map each gate in an XMG into a quantum net-
work and then compose these networks [9]. As quantum computers
can only implement reversible computations, auxiliary qubits are
used to store intermediate results. Besides, the number of T gate is
measured as the cost of a quantum network. Given the results pro-
duced in [9] over the integer reciprocal design INTDIV(n) for n =
16, 32, 64, and 128, we use the proposed SR to optimize XMG. The
results shown in Table 2 indicate our method can further optimize
the T gate count by 6% with additional 5% qubits. Since the T gate
count accounts for the far most complex execution in a quantum
computer [5], the optimization of T gate count has a significant
impact on quantum circuit realization.

6.4 Discussions
Although the SR method can optimize XMG size, due to the de-
ferred transformation rules can only be implemented in two logic
levels, the optimization results have space for further improvement.
There are more optimization obstacles of heterogenous logic rep-
resentation than the homogenous ones. As next steps, we aim to
integrate both FR and SR to exploit more robust XMG size optimiza-
tion method. Moreover, as ABC technology mapper f1 is based on
AIGs instead of MIGs or XMGS, both FR and SR results may be
improved if an MIG-based technology mapper is developed. The
results demonstrated show the XMG quality improvements diminish
after technology mapping; the mapping-aware structural rewriting
is also of high interest.

7 CONCLUSIONS
In this paper, we proposed a structural rewriting method for
XOR-Majority Graphs. By exploiting XOR and majority Boolean
logic identities, we found the optimization opportunities during
XMG structural rewriting. By evaluating EPFL and ISCAS bench-
marks, the experimental results show we can further optimize XMG
size/depth product by 32% and LUT size/depth product by 5% by
giving an functional rewriting XMG as a starting point. The pro-
posed method is also applied for quantum circuit synthesis, which
reduce the T gate count by 6%.

ACKNOWLEDGMENTS
The authors would like to thank anonymous reviewers for useful
comments. The work was partly supported by the National Natural
Science Foundation of China under Grant Nos.: 61501268, 61871242
and 61471211 and by the Swiss National Science Foundation under
Grant No.: 200021-169064 MAJesty.

Table 2: Experimental Results on Quantum Circuit Realiza-
tion of Reciprocal Operation (INTDIV(n))

<table>
<thead>
<tr>
<th>n</th>
<th>Before rewriting [9]</th>
<th>After XMG rewriting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>qubits</td>
<td>T-count</td>
</tr>
<tr>
<td>16</td>
<td>429</td>
<td>14740</td>
</tr>
<tr>
<td>32</td>
<td>1705</td>
<td>59066</td>
</tr>
<tr>
<td>64</td>
<td>6810</td>
<td>238182</td>
</tr>
<tr>
<td>128</td>
<td>26688</td>
<td>921368</td>
</tr>
<tr>
<td>Avg.</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

REFERENCES
Majority-inverter graph: A novel data-structure and algorithms for efficient
logic optimization. In DAC. 1–6.
Majority-Inverter Graph: A new paradigm for logic optimization. IEEE Trans on
Technical Report. FMV Reports Series, Institute for Formal Models and Verifica-
tion, Johannes Kepler University, Altenbergerstr 69, 4040 Linz, Austria.
circuits and a target translator into FORTRAN. In ISCAS. 659–662.
and Giovanni De Micheli. 2017. A novel basis for logic rewriting. In ASP-DAC.
151–156.
necessary?. In DDECS. 134–139.
to area optimization for LUT-based technology mapping. In FPGA. 147–156.
Hill Higher Education.
vanni De Micheli. 2017. Exact synthesis of majority-inverter graphs and its
applications. IEEE Transactions on Computer-Aided Design of Integrated Circuits
Design automation and design space exploration for quantum computers. In DATE.
470–475.