Inversion Minimization in Majority-Inverter Graphs

Eleonora Testa[1], Mathias Soeken[1], Luca Gaetano Amaru[2], Pierre-Emmanuel Gaillardon[3], Giovanni De Micheli[1]
[1] EPFL, Switzerland [2] Synopsys, CA, USA [3] University of Utah, UT, USA
eleonora.testa@epfl.ch

***Abstract*—In this paper, we present new optimization techniques for the recently introduced *Majority-Inverter Graph* (MIG). Our optimizations exploit intrinsic algebraic properties of MIGs and aim at rewriting the complemented edges of the graph without changing its shape. Two exact algorithms are proposed to minimize the number of complemented edges in the graph. The former is a dynamic programming method for trees; the latter finds the exact solution with a minimum number of inversions using *Boolean satisfiability* (SAT). We also describe a heuristic rule based algorithm to minimize complemented edges using local transformations. Experimental results for the exact algorithm to fanout-free regions show an average reduction of 12.8% on the EPFL benchmark suite. Applying the heuristic method on the same instances leads to a total improvement of 60.2%.**

I. Introduction

Nanotechnologies are recently being studied as replacement or enhancement for CMOS. Devices in these nanotechnologies have logic models different from standard transistors and many of them realize majority gates as primitive building blocks. Examples of these nanotechnologies are *Quantum-dot Cellular Automata* (QCA, [1], [2]), *Spin Wave Devices* (SWD, [3], [4]), *Spin-Transfer-Torque* Devices (STT, [5]), *Resistive Random Access Memories* (RRAMs, [6], [7], [8]). To properly assess these post-CMOS technologies, *Electronic Design Automation* (EDA) tools necessitate new logic synthesis techniques and abstractions [9]. Much work concerning majority synthesis has been carried out back in the 1960s [10], [11]. Recently, *Majority-Inverter Graphs* (MIG, [12]) are found to suitably abstract novel majority based nanotechnologies [13], [14], besides being a useful tool to reduce area and delay in standard CMOS circuits. MIGs use the majority-of-three function $\langle xyz \rangle = xy \vee xz \vee yz$ and negation as only logic primitives; negations are simply represented as complemented edges in the graph, similarly as in BDDs.

Inversion minimization plays a predominant role in emerging technologies whose circuits are built using only majorities (MAJ) and inverters (INV). Area and delay costs depends on the number of INVs in the circuit; for instance, a QCA majority gate and inverter are presented in [2]; the majority gate requires five QCA cells, while the inverter gate requires up to 13 QCA cells.

Previous work has considered inversion minimization [15]. In this work, we exploit the intrinsic algebraic properties of MIGs which allow for superior rewriting possibilities as compared to other logic representations such as *And-Inverter Graphs* (AIGs). As an example, negations can be freely propagated through an MIG using self-duality, i.e., $\langle \bar{x}\bar{y}\bar{z} \rangle = \overline{\langle xyz \rangle}$.

In this paper, we present MIG rewriting techniques that target at optimizing inversions within the logic network. The core idea is to minimize the number of complemented edges in the graph without changing its shape. The shape is the way in which all the nodes are connected through edges by disregarding complemented edges. We present two exact algorithms to minimize inversions in the MIG. First, we focus on a dynamic programming algorithm for trees; then we propose a minimization method based on *Boolean Satisfiability* (SAT). These methods show good results, but the former is optimal for fanout-free MIGs, while the latter can be applied to small graphs only. We also describe heuristic approaches to obtain a local minimum in the number of complemented egdes. Our experimental results show that a reduction of 60.2% is achieved on average.

The paper is organized as follows. Section II introduces background on MIG; Section III focuses on the two exact approaches and on the heuristic method used to minimize complemented edges. Section IV shows experimental results and Section V concludes the paper.

II. Background

In this section, we describe MIGs [12], [16], [17], which are logic representation forms based on majority logic. A MIG is a data structure for Boolean function representation and optimization. It is defined as a homogeneous logic network consisting of 3-input majority nodes and regular/complemented edges.

MIGs can efficiently represent Boolean functions thanks to the expressive power of the majority operator. Indeed, a majority operator can be configured to behave as a traditional conjunction (AND) or disjunction (OR) operator. In the case of 3-input majority operator, fixing one input to `0` realizes an AND while fixing one input to `1` realizes an OR. As a consequence of the AND/OR inclusion by MAJ, traditional *AND/OR/INV Graphs* (AOIGs) are a special case of MIGs and MIGs can be easily derived from AOIGs. An example of MIG representations derived from its optimal AOIG is depicted by Fig. 1a. AND/OR operators are replaced node-wise by MAJ-3 operators with a constant input.

Intuitively, MIGs are at least as compact as AOIGs. However, even smaller MIG representation arises when fully exploiting the majority functionality, i.e., with non-constant inputs [12].

We are interested in compact MIG representations because they translate into smaller and faster physical implementations. In order to manipulate MIGs and reach advantageous MIG

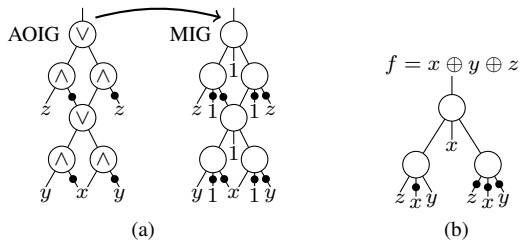

Fig. 1. Example (a) of MIG representations (right) for $f = x \oplus y \oplus z$ derived by transposing its optimal AOIG representations (left). Complement attributes are represented by bubbles on the edges. Example (b) of optimized MIG for f.

representations, a dedicated Boolean algebra was introduced in [16]. The axiomatic system for the MIG Boolean algebra, referred to as Ω, is defined by the five following primitive transformation rules.

$$\Omega \begin{cases} \textbf{Commutativity} - \boldsymbol{\Omega.C} \\ \langle xyz\rangle = \langle yxz\rangle = \langle zyx\rangle \\ \textbf{Majority} - \boldsymbol{\Omega.M} \\ \langle xxy\rangle = x \qquad \langle x\bar{x}y\rangle = y \\ \textbf{Associativity} - \boldsymbol{\Omega.A} \\ \langle xu\langle yuz\rangle\rangle = \langle zu\langle yux\rangle\rangle \\ \textbf{Distributivity} - \boldsymbol{\Omega.D} \\ \langle xy\langle uvz\rangle\rangle = \langle\langle xyu\rangle\langle xyv\rangle z\rangle \\ \textbf{Inverter Propagation} - \boldsymbol{\Omega.I} \\ \overline{\langle xyz\rangle} = \langle \bar{x}\bar{y}\bar{z}\rangle \end{cases} \quad (1)$$

Some of these axioms are inspired from median algebra and others from the properties of the median operator in a distributive lattice. A strong property of MIGs and their algebraic framework is about reachability. It has been proven that, by using a sequence of transformations drawn from Ω, it is possible to traverse the entire MIG representation space [12]. In other words, given any two equivalent MIG representations, it is possible to transform one into the other by just using axioms in Ω. This results is of paramount interest to logic synthesis because it guarantees that the best MIG, for a given target metric, can always be reached. Unfortunately, deriving such an optimal sequence of transformations is an intractable problem. As for traditional logic optimization, heuristic techniques provide here fast solutions with reasonable quality [18]. An efficient depth optimization heuristic has been introduced that iterates local Ω rules over the critical path in order to push up variables with late arrival times.

As previously anticipated, by using the MIG algebraic framework, it is possible to obtain better MIGs for the example in Fig. 1a. Fig. 1b shows the new MIG structure, which is optimized in both depth (number of levels) and size (number of nodes). These MIG can be reached using a sequence of Ω axioms starting from their unoptimized structures. We refer the reader to paper [12] for an in-depth discussion on MIG optimization recipes.

III. Inversion Minimization

This section describes the techniques employed to minimize the number of complemented edges. First, we present two exact algorithms to minimize inversions in the graph. Then, we focus on a heuristic minimization method based on the local manipulation of each node using the axiom $\Omega.I$.

A. Tree based Exact Algorithm

We first present an exact algorithm to minimize complemented edges for the case when the MIG is a tree, i.e., if every node has at most one output. Here, inversion minimization is possible using the axiom $\Omega.I$. The axiom $\overline{\langle xyz\rangle} = \langle \bar{x}\bar{y}\bar{z}\rangle$ states that the complemented edges can be moved without neither changing the number of nodes nor the edges connections in the graph. For each node, the complemented edges configuration includes three input edges and one output edge; the original configuration is the configuration of the node in the MIG, while the changed configuration is the configuration of the node changed according to the axiom $\Omega.I$. The cost of a node is the sum of the number of complemented edges on three children and the cost of each children. Complements on constant inputs are not accounted in the cost, since in physical implementations both constants `0` and `1` are available.

The presented method uses dynamic programming and computes best configurations in a topological order. We notice (i) that the cost of complemented edges depends on the cost of the three children, and (ii) that two optimum configurations for a node have different effects on the parent node only if they show opposite polarities for the node's output.

The algorithm is applied to the nodes going from the inputs (leaves) to the outputs (roots) of the graph. Since we traverse the nodes in topological order, for every node v, we know the minimum complemented edges configuration for each children and the cost of each configuration.

The algorithm considers two cases depending on which level the node belongs:

(i) nodes with all children being primary inputs,
(ii) all other nodes.

Case (i): We save both the original configuration of complemented edges and the one changed according to $\Omega.I$. These two configurations have different output polarities and then they cause a different effect on the parent node. The two configurations and their costs are stored.

Case (ii): At this point, we have computed at most two optimum configurations for each child from which we can compute at most 8 configurations for the node. From these the best configuration for each output polarity is saved. If only one output polarity is covered by all configurations, we apply $\Omega.I$ on the computed configurations and pick the best one from the newly generated ones.

It is worth noticing that for the output nodes only one configuration is saved. The configuration with the smallest cost is saved independently of the output polarity; if two configurations have the same cost, only one is saved since there is no effect on the MIG.

The algorithm is explained in Alg. 1. An MIG $M = (V, E, Y)$ is a DAG consisting of a finite set of nodes V, a finite multiset of edges E and a finite multiset of outputs Y [19]. The MIG resulting from the algorithm is a new MIG $\hat{M}$ with a different complemented edges configuration. The nodes are evaluated in topological order and depending on their level. For nodes v with all children being primary inputs, two configurations are saved, v and $\hat{v}$. For other nodes, all the combinations of children configurations are considered. W is a finite set including all configurations of node v, while $\hat{W}$ consists of configurations changed according to the axiom. Two configurations are saved, which respectively are the best one in W and $\hat{W}$. If v is an output node, the configuration with the lower cost is add to $\hat{M}$.

Data: MIG $M = (V, E, Y)$
Result: Optimized MIG $\hat{M}$
foreach $v \in$ topsort(V) **do**
  if *all children of* v *are primary inputs* **then**
    set $\hat{v} \leftarrow \Omega.I(v)$;
    set conf$(v) \leftarrow \{v, \hat{v}\}$;
  else
    let v_1, v_2, v_3 be the children of v;
    set $W \leftarrow \{\}$, $\hat{W} \leftarrow \{\}$;
    foreach *combination* v' *of* conf(v_1), conf(v_2), conf(v_3) **do**
      set $W \leftarrow W \cup \{v'\}$;
      set $\hat{W} \leftarrow \hat{W} \cup \{\Omega.I(v')\}$;
    end
    set conf$(v) \leftarrow \{\min W, \min \hat{W}\}$;
    if v *is an output node* **then**
      add the best configuration in conf(v) to $\hat{M}$.
    end
  end
end

Algorithm 1: Complemented edges minimization in a tree.

Example 1: Alg. 1 is applied to the MIG in Fig. 2a. We will use the symbol $\hat{}$ for nodes changed according to axiom $\Omega.I$. First, nodes 1 and 2 are considered since their inputs are primary inputs only. For these nodes, two configurations are saved. The costs of the nodes are evaluated and stored. For instance, for node 1 the cost is equal to 2, while for node $\hat{1}$ the cost is 1, since only one input is complemented. Four configurations are possible for node 3, since two of its children (1 and 2) have two configurations stored. They are shown in Fig. 3a. The costs are given by the sum of cost of node 3 and the costs of its children. Four configurations are possible for node $\hat{3}$ and they are shown in Fig. 3b. All the configurations of node 3 have an opposite polarity with respect to node $\hat{3}$. Two configurations with opposite polarities need to be saved. We save the ones with the smallest cost for node 3 and node $\hat{3}$, which respectively are configuration (i) for node 3 and (ii) for $\hat{3}$. For node 4 and $\hat{4}$, two combinations are present. Since 4 is an output node, only the configuration with the smallest cost is saved.

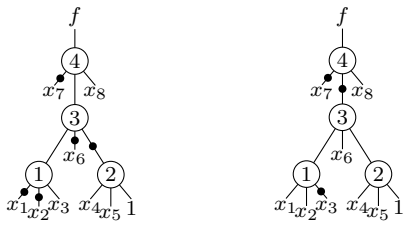

Fig. 2. Example: MIG for function f, before (a) and after (b) minimization of complemented edges using the exact algorithm for trees.

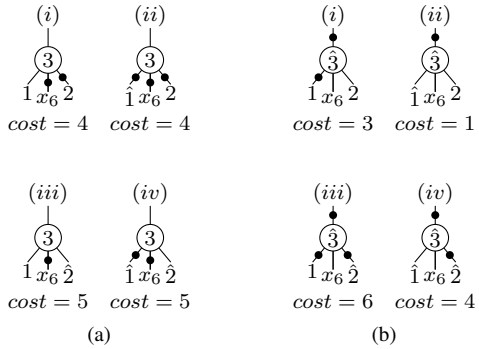

Fig. 3. All combinations for node 3 (a) and node $\hat{3}$ (b).

The final MIG is shown in Fig. 2b. The total amount of complemented edges is reduced from 5 to 3.

B. SAT based Exact Algorithm

Here, we aim at minimizing the number of complemented edges using a SAT based approach. Given an MIG, the Boolean function f that it represents, and the number of complemented edges p, the purpose is to find the minimum number of complemented edges $p_{\min}$ for the given graph. In order to be consistent with the previous algorithm, the shape and the number of nodes of the graph are not changed by this method. It is worth noticing that the SAT based algorithm can be adopted to find an MIG with $p_{\min}$ and a different number of nodes and shape.

Our approach finds optimum solutions by solving decision problems that ask whether there exists an MIG in which the number of complemented edges is $p_{\min}$. The algorithm used and the encoding of the problem are similar to the ones proposed in [19]. The instance is a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ represented by the MIG, which has k nodes and p complemented edges. To find the solution, we start by solving the decision problem for $p_{\min} = p$ and decrease the value until the solution becomes unsatisfiable. According to [19], the number of nodes is equal to k and each node with index $l \in \{1, ..., k\}$ can be represented with 10 variables:

- three inputs $a_{1,l}^{(j)}, a_{2,l}^{(j)}, a_{3,l}^{(j)} \in \mathbb{B}$ of gate l,
- one output $b_l^{(j)} \in \mathbb{B}$ of gate l,
- three select variables $s_{1,l}, s_{2,l}, s_{3,l} \in \mathbb{B}^{[\log_2(n+l)]}$ that encode which are the children of gate l, and

- polarity variables $p_{1,l}, p_{2,l}, p_{3,l} \in \mathbb{B}$ that represented regular/complemented edges of the children.

Variable j ranges from 0 to $2^n - 1$ and each j represents one input assignment in f; hence each node is duplicated 2^n times.

Regarding the problem constraints, the *Majority Functionality* and the *Function Semantics* are ensured by constraints proposed in [19]. Two more constraints are considered for this problem: (i) one on the select variables in order to have the same MIG structure as the original one and (ii) one on the polarities variables for the minimum number of complemented edges. These two constraints are discussed below.

1) Input Connections: For this decision problem, the number of nodes k is given and it is equal to the number of nodes of the original MIG. Since we want to have the same structure for the MIG, the values for the three select variables $s_{1,l}, s_{2,l}, s_{3,l}$ are known and are taken from the MIG. For each node, since all its children belong to a lower level, the values for the three select variables range from 0 to $n + (l - 1)$, where n is the number of inputs. If $s_{c,l}$ is equal to 0, it means that the constant value 0 is child c of node l. If the input of a node is a primary input, then its select variable is lower than or equal to n; while a value between n and $n + (l - 1)$ ensures a connection with an internal node in the lower level.

2) Minimum Complemented Edges: To make sure that the number of complemented edges equal to $p_{\min}$, some clauses are added to the problem. A SAT encoding for cardinality constraints such as $x_1 + \cdots + x_m \leq r$ is proposed in [20]. We add similar clauses to ensure that the sum of all the polarities variables and the polarity of the output is equal to $p_{\min}$. It is worth noticing that also in this algorithm, complemented edges that point to constant 0 are not considered in the sum. The number m of polarity variables is $3k + 1 - z$, where k is the number of nodes of the MIG and z is the number of constant inputs. For this case, $r = p_{\min}$. We consider $(m - r) \cdot r$ new variables q_v^h with $1 \leq v \leq (m - r)$ and $1 \leq h \leq r$, and new clauses for the SAT problem:

$$\bar{q}_v^h \vee q_{v+1}^h, \text{ for } 1 \leq v < (m - r) \text{ and } 1 \leq h \leq r, \quad (2)$$

$$\bar{x}_{v+h} \vee \bar{q}_v^h \vee q_v^{h+1}, \text{ for } 1 \leq v \leq (m - r) \text{ and } 0 \leq h \leq r, \quad (3)$$

where $\bar{q}_v^h$ is not added for $h = 0$ and q_v^{h+1} is omitted when $h = r$.

The decision problem is solved using the SMT solver Z3 [21]. Fig. 4a is given in order to explain the approach used. It shows an MIG in which the initial number of complemented edges is $p = 6$ and the number of nodes is $k = 3$. The select variables are:

$$s_{1,1} = 1, s_{2,1} = 2, s_{3,1} = 3; \quad (4)$$

$$s_{1,2} = 7, s_{2,2} = 5, s_{3,2} = 4; \quad (5)$$

$$s_{1,3} = 6, s_{2,3} = 7, s_{3,3} = 8. \quad (6)$$

The problem we want to solve is

$$p_{1,1}+p_{2,1}+p_{3,1}+p_{1,2}+p_{2,2}+p_{3,2}+p_{1,3}+p_{2,3}+p_{3,3}+y \leq p,$$

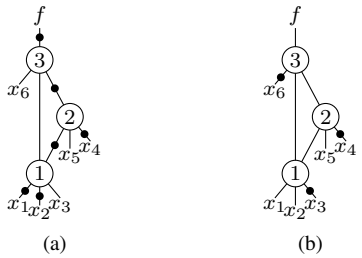

Fig. 4. Example: MIG for function f, before (a) and after (b) minimization of complemented edges using a SAT-solver based approach.

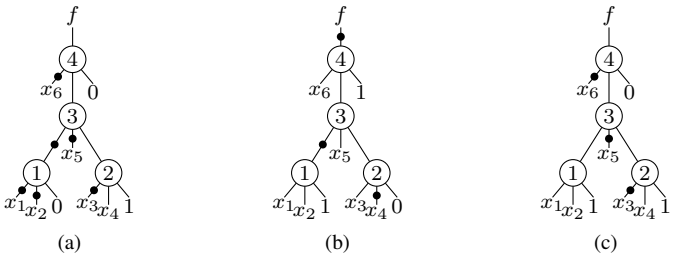

Fig. 5. Another example: MIG for function f (a); minimization of complemented edges using the tree method (b) and a SAT-solver based approach (c).

where y is the output polarity according to [19]. To find the minimum number of complemented edges, we decrease p until the problem becomes unsatisfiable and $p_{\min}$ is found. For the example in Fig. 4, the minimum number of complemented edges is 3. The final result is shown in Fig. 4b. The number of nodes and the structure of the MIG are the same as the input graph.

As shown by the previous example, this approach is not limited to fanout-free circuits as the method described in Section III-A. Fig. 5 shows a circuit optimized with both the tree method (Fig. 5b) and the SAT based method (Fig. 5c). The resulting circuit has a different complemented edges configuration, but the minimum number is the same for both methods.

This SAT-oriented approach allows us to find the exact solution in terms of number of complemented egdes; but it does not scale for large functions.

C. Heuristic Algorithm by Local Rewriting

Both the exact approaches described in previous section suffer from some limits: the former can be applied to fanout-free graphs only, while the second is perfect for graphs of limited size. Here, we propose a heuristic method to minimize complemented edges. We aim at minimizing the complemented edges in the MIG by recursively applying the inverter propagation axiom $\Omega.I$, given by $\overline{\langle xyz \rangle} = \langle \bar{x}\bar{y}\bar{z} \rangle$.

The main idea to minimize complemented edges is to use $\Omega.I$ to move complemented edges on the inputs to the output. To reduce the number of complemented edges, we apply the transformations rules mentioned below on all the nodes of the MIG. These transformations do not change the depth nor

Fig. 6. MIG for a full adder, before (a) and after (b) one-level rules to decrease the number of complemented edges. Bubbles represent complementation of the edges

the size of the MIG. Different ways of applying $\Omega.I$ can be adopted depending on whether the node has constant inputs or not. Furthermore, the axiom can be applied considering one node at a time (one-level) or evaluating savings in the number of complemented edges for two levels of the MIG (two-level). A taxonomy of the rules used to decrease complemented edges is proposed in the following; then the procedure used to minimize complemented edges is described.

1) One-level Rules for MAJ: Here, we describe rules that apply to *MAJ* nodes, which are nodes with no constant inputs. The rules used to decrease complemented edges aim at moving complemented edges of the inputs to the output. They can be formalized as:

$$\text{Rules} \begin{cases} \textbf{MAJ\_3} \\ \begin{cases} \langle \bar{x}\bar{y}\bar{z}\rangle = \overline{\langle xyz\rangle} \\ \overline{\langle \bar{x}\bar{y}\bar{z}\rangle} = \langle xyz\rangle \end{cases} \\ \textbf{MAJ\_2} \\ \overline{\langle x\bar{y}\bar{z}\rangle} = \langle \bar{x}yz\rangle \end{cases} \tag{7}$$

The MAJ_3 rules consider nodes in which the three inputs are complemented. Considering each node, these transformations lead to a decrease in the number of complemented edges equal to:

$$3 + (\#\text{CO} - \#\text{NCO}) \tag{8}$$

where $\#$CO is the number of complemented outputs of the node and $\#$NCO are the uncomplemented outputs. Savings for the MAJ_2 rule are equal to:

$$1 + (\#\text{CO} - \#\text{NCO}) \tag{9}$$

For these rules, only one node is considered at a time. We evaluate savings according to the formulas mentioned above; each node is changed using one of the transformation rules if savings larger than 0 can be achieved. An example is given in Fig. 6. Fig. 6a shows the MIG of the full adder composed by the three nodes 1, 2 and 3. It is possible to apply the MAJ_3 rule on node 2 with savings of 3 and the MAJ_2 on node 1 with savings of 2. Fig. 6b illustrates the MIG of the full adder after the one-level transformations applied on the nodes of the first level. In this example, the number of complemented edges is reduced from 7 to 2.

2) One-level Rules for AND/OR: All the rules mentioned above are applied to nodes in which none of the inputs is set to a constant value. It is worth noticing that they equally work for *AND* and *OR* nodes, which are majority nodes in which one of the input is set to `0` and `1`, respectively. Our MIG data structure only has constant `1`, hence in AND nodes the constant input is $\bar{1} = 0$. The rules used to decrease complemented edges for AND/OR are:

$$\text{AND/OR Rules} \begin{cases} \textbf{AND\_3} \\ \begin{cases} \langle \bar{x}\bar{y}\bar{1}\rangle = \overline{\langle xy1\rangle} \\ \overline{\langle \bar{x}\bar{y}\bar{1}\rangle} = \langle xy1\rangle \end{cases} \\ \textbf{AND\_2} \\ \overline{\langle x\bar{y}\bar{1}\rangle} = \langle \bar{x}y1\rangle \\ \textbf{OR\_2} \\ \overline{\langle \bar{x}\bar{y}1\rangle} = \langle xy\bar{1}\rangle \end{cases} \tag{10}$$

Note that, as in the previous algorithms, complements on constant inputs are not accounted in the total amount of complemented edges. The savings estimation is adjusted for the AND/OR cases. The AND_3 rule has savings equal to:

$$2 + (\#\text{CO} - \#\text{NCO}) \tag{11}$$

while for the AND_2 rule they are equal to:

$$\#\text{CO} - \#\text{NCO} \tag{12}$$

For OR rule savings are equal to:

$$2 + (\#\text{CO} - \#\text{NCO}) \tag{13}$$

3) Two-level Rules: In two level transformations, we consider one node (main node) and all the nodes on its outputs (parents). By doing this, we account that transforming only the main node may not result in savings greater than 0, but this transformation changes the complemented edges pattern of the parents and consequently may result in a total two-level savings larger than 0 when applying transformations on the parents. We evaluate the total two-level savings as the sum of the savings obtained by changing the main node and the savings achieved on the parents if the main node is changed. If the total two-level savings are positive, first the main node is changed according to the rules (7), then all the one-level rules are applied to the parents. In Fig. 7a, none of the nodes, which are called here node 1, 2 and 3, can be changed by the one-level transformation since there is no benefit in the total number of complemented edges. On the other hand, changing node 1 according to the inverter propagation axiom generates the possibility of applying one of the one level rules on node 3. If node 1 is changed, its savings are equal to 0, but, thanks to this first transformation, the MAJ_2 rule can be applied to node 3. Fig. 7b shows the full adder after changing node 1; while Fig. 7c shows the final circuit. In this case, the number of complemented edges is not reduced from step a to step b but it is reduced from step b to c, with total savings equal to 2.

4) Minimization Procedure: We minimize the complemented edges using the reduction rules proposed above in the way given in Alg. 2. We describe a possible order of applying the rules, but they can be applied in a different way producing different results. When the number of complemented edges before optimization is equal to the number of complemented

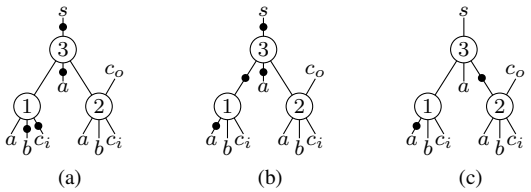

Fig. 7. Two-level transformation applied on the MIG of a full adder.

edges after the optimization, it means that no further rules can be applied to the MIG and we have reached the minimal number of complemented edges achievable with our procedure. We apply each single rule from node 0 to the last node of the circuit. At the end of the minimization process, the MIG has the same logical depth and the same size of the unoptimized one, since all transformation rules do not change the shape of the graph but aim at changing the number of complemented edges.

```
while the number of compl. edges is decreasing do
    AND_3(n) for each node n;
    AND_2(n) for each node n;
    MAJ_3(n) for each node n;
    MAJ_2(n) for each node n;
    Two-level(n) for each node n;
end
```

Algorithm 2: Heuristic algorithm to minimize complemented edges.

IV. Experimental Results

In this section, we present results obtained in terms of minimization of complemented edges. First, we test the tree algorithms on the fanout-free regions of the arithmetic circuits of the EPFL benchmarks,[1] then we describe the further improvement obtained on the same instances with the heuristic approach and the minimization rules proposed in Section III-C. Preliminary experimental results are provided for the SAT based approach.

A. Tree based Exact Algorithm on Fanout-free Regions

We developed a C program to implement the exact algorithm for trees proposed in III-A. The algorithm has been applied to the fanout-free regions of instances which are optimized using the rewrite recipe described in [12]. The fanout-free regions are evaluated in topological order and the nodes with more than one output are not changed by the algorithm.

We compare the number of complemented edges; results are shown in Table I.

The size and the depth of the MIG are the same before and after minimization. The improvement in the number of complemented edges is 12.8% on average. Table I lists the number of fanout-free regions and their average size for each instance. As expected, the larger improvement in the number of complemented edges is for the instance in which the fanout-free regions have larger size (max). Since, on average, the fanout-free regions have small size, an optimization across fanout-free regions boundaries is required.

[1] http://lsi.epfl.ch/benchmarks

TABLE I
Exact tree algorithm on fanout-free regions

Benchmark	$\#N$	#FFR	S_{ffr}	#CE	$\#CE_{tree}$	Impr.
adder	2978	1447	2.1	2905	2532	12.8%
divisor	75666	28058	2.7	78302	67622	13.6%
log2	37582	13374	2.8	38585	33877	12.2%
max	7202	1421	5.1	6543	4778	27.0%
multiplier	41885	14886	2.8	39589	37530	5.2%
sin	7890	3002	2.6	7625	6642	12.9%
sqrt	52344	19697	2.7	53327	44788	16.0%
square	19200	11277	1.7	23390	22752	2.7%
Average			2.8			12.8%

$\#N$: number of MIG nodes, #FFR: number of fanout-free regions, S_{ffr}: average size of the fanout-free regions, #CE: number of complemented edges, $\#CE_{tree}$: number of complemented edges after minimization with the exact tree algorithm on the fanout-free regions, Impr: improvement with regards to #CE.

TABLE II
Heuristic Algorithm by local rewriting

Benchmark	#CE	$\#CE_{tree}$	$\#CE_{local}$	Impr.	Tot_Impr.
adder	2905	2532	1192	52.9%	59.0%
divisor	78302	67622	30748	54.5%	60.7%
log2	38585	33877	14367	57.6%	62.8%
max	6543	4778	3464	27.5%	47.1%
multiplier	39589	37530	18829	49.8%	52.4%
sin	7625	6642	3017	54.6%	60.4%
sqrt	53327	44788	19889	55.6%	62.7%
square	23390	22752	5404	76.2%	76.9%
Average				53.6%	60.2%

#CE: number of complemented edges, $\#CE_{tree}$: number of complemented edges obtained in IV-A, $\#CE_{local}$: number of complemented edges after minimization with heuristic algorithm by local rewriting, Impr: improvement with regards to $\#CE_{tree}$, Tot_Impr: improvement with regards to #CE.

B. Heuristic Algorithm by Local Rewriting

To overcome limits due to the reduced size of fanout-free regions, we developed a C program to implement the heuristic algorithm proposed in Section III-C. We apply the algorithm on instances in which the number of complemented edges is already reduced with the method discussed in Section III-A. The code implements all the rules described in Section III-C. The rules are applied as proposed in Alg. 2 on all the nodes of the MIG, from the first one to the last one. A loop iterates the process until the minimal number of complemented edges is reached. The code changes the number of complemented edges of the MIG without transforming the shape of the graph. Results are shown in Table II.

The largest improvement in the number of complemented edges is obtained after the first loop. On average, 3.5 loops are necessary to reach the minimal number of complemented edges. The algorithm is applied on instances in which the number of complemented edges has been previously decreased

with the tree algorithm on fanout-free regions. The further improvement is of 53.6% on average. This heuristic approach allows a total improvement in the number of complemented edges equal to 60.2% on average.

C. SAT based Algorithm

We have run the exact synthesis algorithm on all unique (fully DSD decomposable) 6 variable functions obtained using structural cut enumeration on all instances of the MCNC, ISCAS, and ITC benchmarks as proposed in [22]. That is, first an MIG with an optimum number of nodes is found which is then resynthesized to achieve the optimum number of complemented edges in a second step. 40195 functions are evaluated and a timeout of 1 minute is given for each function. Among these 40195, we encountered 2264 timeouts and in the remaining ones, we achieved a reduction in the number of complemented edges of 30.8% in average, requiring 0.3 seconds. In the best case, the complemented edges are reduced by 83.3% (from 12 to 2) with a runtime equal to 0.7 seconds.

V. Conclusion

We described two exact algorithms to minimize the number of complemented edges in a MIG. The tree exact algorithm can be applied to MIGs in which each node has one output only, while the SAT based approach is suitable for MIGs of limited size. We illustrated also a heuristic method to decrease complemented edges.

At the logic level, our results showed that a decrease in the number of complemented edges of 12.8% on average can be achieved when applying the tree exact algorithm on the fanout-free regions of the circuits. Experiments showed that due to the limited size of fanout-free regions, an optimization that considers fanout-free regions boundaries is needed. Experimental results illustrated that a total improvement of 60.2% on average can be reached using the heuristic method on instances previously optimized using the tree algorithm on fanout-free regions.

The preliminary experimental evaluation for the SAT based algorithm shows promising results. Possible directions for future works include (i) an alternative encoding of the SAT based algorithm using a smaller number of variables and (ii) a peephole optimization of small subnetworks in large MIGs using the SAT based method.

Acknowledgment

This research was supported by H2020-ERC-2014-ADG 669354 CyberCare.

References

[1] I. Amlani, "Digital logic gate using quantum-dot cellular automata," *Science*, vol. 284, no. 5412, pp. 289–291, 1999.

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

[3] T. Schneider, A. A. Serga, B. Leven, B. Hillebrands, R. L. Stamps, and M. P. Kostylev, "Realization of spin-wave logic gates," *Applied Physics Letters*, vol. 92, no. 2, pp. 1–4, 2008.

[4] O. Zografos, L. Amarú, P.-E. Gaillardon, P. Raghavan, and G. De Micheli, "Majority logic synthesis for spin wave technology," *Conference On Digital System Design*, pp. 691–694, 2014.

[5] Z. Pajouhi, S. Venkataramani, K. Yogendra, A. Raghunathan, and K. Roy, "Exploring spin-transfer-torque devices for logic applications," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. PP, no. 99, p. 1, 2015.

[6] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature materials*, vol. 9, no. 5, pp. 403–6, 2010.

[7] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (plim) computer," in *Design Automation and Test in Europe*, 2016.

[8] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs," *Design Automation and Test in Europe*, 2016.

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

[10] S. B. Akers, "Synthesis of combinational logic using three-input majority gates," *Annual Symposium on Switching Circuit Theory and Logical Design*, pp. 149–158, 1962.

[11] R. Lindaman, "A theorem for deriving majority-logic networks within an augmented Boolean algebra," *IRE Trans. On Electronic Computers*, vol. 47, pp. 338–342, 1960.

[12] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: a new paradigm for logic optimization," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–14, 2015.

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

[14] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-based synthesis for nanotechnologies," *Asia and South Pacific Design Automation Conference*, pp. 499–502, 2016.

[15] S. Muroga, *Threshold logic and its applications*. NY, New York: John Wiley & Sons Inc., 1971.

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

[17] ——, "Boolean logic optimization in majority-inverter graphs," *Design Automation Conference*, 2015.

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

[19] M. Soeken, L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Optimizing majority-inverter graphs with functional hashing," *Design Automation and Test in Europe*, 2016.

[20] C. Sinz, "Towards an optimal CNF encoding of Boolean cardinality constraints," *Conference on Principles and Practice of Constraint Programming*, pp. 827–831, 2005.

[21] N. Bjørner and L. de Moura, "Z3: An efficient SMT solver," *Tools and Algorithms for the Constructions and Analysis of Systems*, pp. 337–340, 2008.

[22] A. Mishchenko, "Enumeration of irredundant circuit structures," *International Workshop on Logic Synthesis*, 2015.