Size Optimization of MIGs with an Application to QCA and STMG Technologies

Heinz Riener[1] Eleonora Testa[1] Luca Amaru[2] Mathias Soeken[1] Giovanni De Micheli[1]
[1]Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
[2]Synopsys Inc., Sunnyvale, CA, USA

ABSTRACT

Majority-inverter graphs (MIGs) are a logic representation with remarkable algebraic and Boolean properties that enable efficient logic optimizations beyond the capabilities of traditional logic representations. Further, since many nano-emerging technologies, such as *quantum-dot cellular automata* (QCA) or *spin torque majority gates* (STMG), are inherently majority-based, MIGs serve as a natural logic representation to map into these technologies. So far, MIG optimization methods predominantly target to reduce the depth of the logic networks, corresponding to low delay implementations in the respective technologies. In this paper, we introduce several methods to optimize the size of MIGs. They can be applied such that the depth of the logic network is preserved; therefore our methods have a direct effect on the physical area, without worsening the delay. Some methods are inspired by existing size optimization algorithms for non-majority-based logic networks, others make explicit use of the majority function and its properties. All methods are Boolean—in contrast to algebraic optimization methods—which has a positive effect on the quality but challenges their implementation. Our experiments show that using our methods the size of MIGs in the EPFL combinational benchmark suite can be reduced by up to 7.12%. When mapped to QCA and STMG technologies we reduce the average area-delay-energy product by 2.31% and 2.07%, respectively.

ACM Reference Format:
Heinz Riener[1] Eleonora Testa[1] Luca Amaru[2] Mathias Soeken[1] Giovanni De Micheli[1]. 2018. Size Optimization of MIGs with an Application to QCA and STMG Technologies. In *NANOARCH '18: IEEE/ACM International Symposium on Nanoscale Architectures, July 17–19, 2018, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3232195.3232202

1 INTRODUCTION

Many of today's nano-emerging technologies, including spin-wave devices [8], quantum-dot cellular automata (QCA, [10]), and spin torque majority gates (STMG, [13]), are inherently majority-based. As an example, the computation principle of spin-wave devices is based on the interference of propagating spin waves and the information is encoded in the phase of the waves. Being majority-based, these technologies offer a particular inexpensive realization of the majority operation. For example, in the QCA technology the area requirements for the majority-of-three operation are more than 2× smaller compared to the ones of an inverter.

The recent progress in nano-emerging technologies have sparked a considerable interest in majority-based logic synthesis. In contrast to conventional logic synthesis algorithms—being based on logic primitives such as 'AND' or 'OR'—majority-based algorithms employ intermediate data-structures capable of natively representing and manipulating majority operations. In particular, competitive solutions for majority-based delay optimization (see, e.g., [1]) and inversion minimization (see, e.g., [18]) have been proposed.

In this paper, we concentrate on majority-based size optimization. We introduce size optimization algorithms for majority-inverter graphs (MIGs), a logic network representation in which the only two primitives are a majority-of-three gate and an inverter. We focus on node replacement techniques that re-express the global function of an existing majority node using other nodes already present in the logic network. Nodes which are no longer used (including nodes in their transitive fan-ins) can then be removed. The objective is to reduce the size of the logic representation as much as possible while maintaining the global input-output functionality of the logic network (and preserving the logic network's depth).

We introduce *Boolean resubstitution* for MIGs, an effective optimization technique in conventional logic optimization flows, which serves two purposes: (1) it achieves size reductions when other techniques saturate and (2) helps to escape local minima in the logic optimization flow and thus re-enables other size optimizations. We show that finding a resubstitution for a single node in an MIG—although requiring in the worst-case cubic time in the number of majority nodes (instead of quadratic time for two-input nodes in conventional logic representations)—can be done efficiently in MIGs by leveraging a novel filtering approach that sorts out infeasible resubstitution candidates as early as possible.

Moreover, we introduce *relevance optimization*, a novel node replacement technique that makes use of the majority operation's functional properties and thus exploits optimization capabilities not easily recognizable in conventional logic representations. We show experimentally that in practice relevance optimization achieves results competitive to general Boolean resubstitution such that logic optimization flows for MIGs benefit from both.

All presented size optimizations can be integrated with existing large-scale logic optimization frameworks. We present a proof-of-concept implementation using windowing in combination with truth tables. Experimental results using the EPFL benchmark suite confirm the effectiveness of the novel optimization techniques. We show that a single optimization pass can reduce the size of an already depth-optimized as well as already size-optimized MIG by

NANOARCH '18, July 17–19, 2018, Athens, Greece

ACM ISBN 978-1-4503-5815-6/18/07…$15.00
https://doi.org/10.1145/3232195.3232202

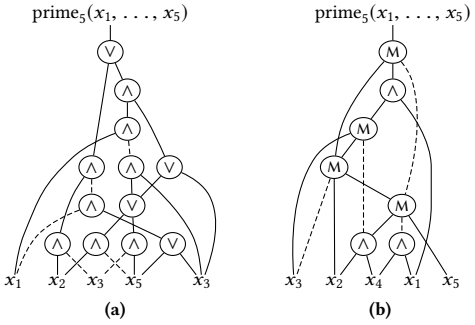

Figure 1: Example of an (a) AOIG representation and (b) MIG representation for function $\text{prime}_5(x_1, \ldots, x_5) = [(x_5 \ldots x_1)_2$ **is prime]. Majority-3, 'AND', and 'OR' nodes are distinguished by** M**,** $\wedge$**, and** $\vee$**, respectively. Complemented edges are drawn using a dashed line.**

up to 7.12% and 2.83% (without increasing its depth[1]), respectively. As baseline, we compare to the best-known state-of-the-art algorithms for MIG depth and MIG size optimization. The resulting improvements are within the expected range of 3% average reduction ratio reported for Boolean resubstitution in conventional logic optimization flows [11] and enable depth- and size-oriented logic optimization flows for MIGs that achieve an average depth reduction of up to 44.17% and an average size reduction of up to 18.13%, respectively.

Almost none of the previous majority-based logic optimization algorithms show the actual gain in area and delay after mapping the logic networks to the corresponding technologies. We demonstrate the effect of the proposed techniques both to the logic minimization of MIGs and to area minimization when mapped to QCA and STMG technologies. The experimental results confirm the effectiveness of our approach. The size of MIGs can be improved by up to 7.12% compared to the state-of-the-art MIG size optimization algorithms. When being mapped to QCA and STMG, the optimized logic networks lead to an improvement in the average area-delay-energy (ADE) product by 2.31% and 2.07%, respectively, compared to the state-of-the-art mapping algorithm.

2 BACKGROUND

2.1 Majority-Inverter Graphs

Majority-inverter graphs (MIGs, [1]) are homogeneous logic networks. A MIG is a directed acyclic graph, where internal nodes represent 3-input majority operations connected via edges that can be complemented to represent inverters. Each node implements the majority function of its three children x, y, and z, denoted as $\langle xyz \rangle$, which evaluates to true if and only if at least two of the three inputs are true [15]. MIGs are universal representation forms, which can be employed to efficiently represent any Boolean function. Traditional AND-OR-inverter graphs (AOIGs) are a special case of MIGs, since $\langle 0xy \rangle = x \wedge y$ and $\langle 1xy \rangle = x \vee y$. It follows that MIGs can be easily derived from AOIGs by node-wise replacement of the 'AND' and 'OR' operators by majority-3 operators with a constant input. Fig. 1 shows both the AOIG and the MIG for the function $\text{prime}_5(x_1, \ldots, x_5) = [(x_5 \ldots x_1)_2$ is prime]. Note that the 'AND' gates in the MIG can be considered as majority nodes where one of its inputs points to constant 0. The example illustrates that MIGs allow for a more compact representation due to the expressiveness of the majority operator. This positively influences the mapping into majority-based technologies.

[1]In fact, the depth is further reduced.

2.2 MIG Optimization Techniques

Besides representing Boolean functions, MIGs allow remarkable logic optimizations. Optimization methods can be classified as algebraic or Boolean and typically aim for either reducing the depth (number of levels) or the size (number of nodes) of an MIG.

Algebraic optimization methods use a sequence of transformation rules to transform an MIG into an optimized version. For this purpose, an MIG Boolean algebra together with its axiomatic system are introduced in [1]. The axiomatic system consists of five primitive axioms (identity, commutativity, distributivity, associativity, and complement) and forms a sound and complete axiomatization of MIG manipulation. As a consequence, given an MIG, all possible functionally equivalent MIG representations can be reached by applying sequences of these axioms. Moroever, three derived rules (relevance, complementary associativity, and substitution) are introduced in [1]. In the context of this paper, only the complement axiom and the relevance rule are important which we introduce as the following transformation rules: (1) The self-duality of the majority operation enables inverter propagation described by the inverter propagation rule $\langle xyz \rangle = \overline{\langle \bar{x}\bar{y}\bar{z} \rangle}$, which allows to move inverters from inputs to outputs, and vice versa. (2) The relevance rule $\langle xyz \rangle = \langle xyz_{x/\bar{y}} \rangle$ allows to replace reconvergent variables with their neighbors, i.e., each occurrence of x in z is replaced by $\bar{y}$ denoted by $z_{x/\bar{y}}$.

Boolean optimization methods, in contrast to algebraic methods, leverage each node's (Boolean) function (and possibly additional don't care information) to improve an MIG representation. In general, Boolean optimization methods are often more precise than algebraic methods and achieve better results, but they are also computationally more costly. One remarkable Boolean method particularly designed for MIGs was proposed in [2]: advantageous orthogonal bit-errors are seeded into the MIG which are automatically corrected leveraging the error correction capabilities of the majority operation.

A logic optimization flow, that employs algebraic and Boolean methods, for MIG depth reduction has been presented in [2].

Work on size reduction of MIGs is more sparse. Recently, an exact synthesis method based on *functional hashing* has been presented [16]. The idea is to rewrite the MIG by replacing all 4-input subgraphs with their minimal-size exact representation. In [16], the search space for 4-input Boolean functions has been reduced by making use of Boolean function classification. Overall, these techniques produce large improvements, but suffer from scalability issues when all Boolean functions with more than 4 inputs have to be precomputed. In [7], the use of exact synthesis for logic rewriting is further improved by computing the exact subgraph only for those functions that appear in practice. This has been achieved by using lookup-table-based mapping techniques (LUT mapping, [14]). In addition to the mentioned optmization methods, other algorithms have been presented. In [9], the network is decomposed into 3-input subgraphs; all 3-input subgraphs are then replaced by their minimal majority expression. Further, node redundancies are removed by

Input: MIG M, cut-size limit l, node limit s
Output: Optimized MIG M

```
foreach node p in M in topological order do
    C = ComputeCut(M, {p}, l);
    W = ExpandToWindow(M, C, s);
    Ŵ = OptimizationProcedure(W, p);
    M = M[W ← Ŵ];
```

Algorithm 1: Windowed MIG Optimization

keeping only one of the nodes implementing the same function. The method has been extended to work with all 4-input subgraphs in [20]. A size optimization *node merging* approach, which removes node redundancies in MIGs, has been presented in [6].

Besides size and depth, other metrics can be optimized. As an example, inversion optimization plays a key role for applications that concern emerging technologies. Some of the most promising nano-technologies face inversion limitations and hence benefit from inversion minimization: (i) some of them do not have an efficient way to implement inversion (see, e.g., QCA [10]); (ii) some others do not have the possibility to build inversions (see, e.g., STMG [12]). In [18], a method to optimize the number of MIG inversions according to the target technology is presented. This method exploits the inverter propagation rule to minimize complemented edges. The same property is used in [19] to remove inversions in the network by moving them on primary inputs.

3 SIZE OPTIMIZATION

In this paper, we revise functional reduction and introduce two new size optimization methods for MIGs: (i) Boolean resubstitution and (ii) relevance optimization. The first method is inspired by existing size optimization algorithms for non-majority-based logic networks; the second method leverages the properties of the majority function. Both methods are Boolean and make use of functional information computed for each node in the logic network. The basis for all optimization methods is the scalable logic synthesis framework described by Mishchenko and Brayton [11]: a small window (with restricted fan-in and unlimited fan-out) is moved over the logic network. The Boolean function of each node within the window is computed using exhaustive simulation. The approach is fast (Boolean functions are represented as truth tables), scales well, and often outperforms computation based on binary decision diagrams [4] or Boolean satisfiability, when windows up to 16 inputs are considered.

3.1 Windowing

Windowing is an approach to limit the scope of an optimization procedure to a small fraction of a logic network that allows in many cases to drastically improve the scalability. The pseudocode of the windowing procedure is shown as Algorithm 1. The procedure takes as input an MIG M and two positive integers l and s, where l denotes the maximal number of primary inputs (cut-size limit) of the window and s denotes the maximal number of nodes (node limit) of the window. As result, the procedure returns the MIG M optimized for size.

In a loop, the procedure iterates over all nodes p of M in topological order and generates for each of the nodes a reconvergence-driven cut C starting from p with at most l nodes (see [11] for a detailed description of the cut computation). The cut C serves as the input boundary of the window W. Starting from the nodes in C, the window W is iteratively extended by merging parent nodes if all their children are already in W. The procedure terminates if no new parents can be merged or the number of window nodes exceeds s. The obtained window W is then locally optimized to $\hat{W}$ using an optimization procedure. Finally, the window W in M is replaced by the optimized window $\hat{W}$.

```
ComputeTruthTable(W);
foreach node u in W in topological order do
    foreach node v in W\{u} in topological order do
        if v ∈ TransitiveFanout(u) then continue;
        if u = v then
            Merge(W, u, v);
        else if u = v̄ then
            Merge(W, u, v̄);
```

Algorithm 2: Functional Reduction

3.2 Functional Reduction

Functional reduction (FR) [5, 6, 9] is an approach that identifies and merges functionally equivalent nodes in a logic network such that after its application no two nodes in the functionally reduced network represent the same logic function. In this section, we revise the basic functional reduction approach of [9] and present a scalable variant utilizing the windowing procedure from the previous section. The pseudocode is shown in Algorithm 2.

Functional reduction is applied to a window W. In an iterative process, each node $u \in W$ is checked for functionally equivalence with each node $v \in W$ not in the transitive fan-out of u. If u and v (u and $\bar{v}$) represent the same logic function, i.e., $u = v$ ($u = \bar{v}$), then u and v ($\bar{v}$) are merged in W, such that the larger logic cone is replaced by the smaller logic cone and the overall size is reduced.

3.3 Boolean Resubstitution

Boolean resubstitution (RS) expresses the logic function of a node using other nodes already present in the logic network. Resubstitution techniques are distinguished by the number k of logic operators additionally added to the logic network when substituting a logic function, i.e., 0-resubstitution expresses a logic function by one other logic function without adding a new logic operator; 1-resubstitution expresses a logic function by adding one logic operator, and so forth.

A resubstitution of a candidate node p with the logic function f is considered beneficial if the number of nodes of W decreases after substitution, i.e., if $\text{Gain}(p, f) \geq 1$ which corresponds to the number of majority operators freed. We consider 0-resubstitution and 1-resubstitution only:

```
ComputeTruthTable(W);
if TryResubstitution0(W, p) then return;
if TryResubstitution1(W, p) then return;
[...]
```

The 0-resubstitution algorithm is an asymmetric variant of functional reduction. Its pseudocode is identical to Algorithm 2, but

```
1  foreach node x ∈ W\{p} in topological order do
2      if x ∈ TransitiveFanout(p) then continue;
3      foreach node y ∈ W\{p, x} in top. order do
4          if y ∈ TransitiveFanout(p) then continue;
5          if p ≠ ⟨xyp⟩ then continue;
6          foreach node z ∈ W\{p, x, y} in top. order do
7              if z ∈ TransitiveFanout(p) then continue;
8              if Gain(p, ⟨xyz⟩) < 1 then continue;
9              if p = ⟨xyz⟩ then
10                 W = W[p ← ⟨xyz⟩];
11                 return true;
12             else if p = ⟨x̄yz⟩ then
13                 W = W[p ← ⟨x̄yz⟩];
14                 return true;
15 return false;
```

Algorithm 3: TryResubstitution1

instead of iterating over all nodes u (line 2) the fixed candidate node p is used.

The 1-resubstitution algorithm shown as Algorithm 3 searches for nodes x, y, z to replace p using one majority operator. Note that due to the inverter propagation rule (see Section 2.2) it suffices to consider $\bar{x}$ as the only negated child. To further speed up computation, we employ a Boolean filter derived from the majority law. If $x \neq y$, then $\langle xyp \rangle = p$ has to hold, i.e., after selecting nodes for x and y, one does not have to iterate over z whenever the filter applies.

3.4 Relevance Optimization

In this section, we introduce relevance optimization (RO), a novel node replacement technique for MIGs that exploits the properties of the majority function and thus cannot be employed when restricted to gate libraries using only 'NOT' and 'AND' or 'NOT' and 'OR'.

Theorem 1 (Replacement rule[2]). *We have $\langle xyz \rangle = \langle wyz \rangle$ if and only if $(x \oplus w)(y \oplus z) = 0$, or in other words $y \neq z \Rightarrow w = x$.*

The replacement rule describes under which condition one operand in a majority expression can be replaced by another one. One can readily verify that the aforementioned relevance rule is a special case of the replacement rule.

Corollary 1 (Relevance rule[2]). *We have $\langle xyz \rangle = \langle x_{y/\bar{z}} yz \rangle$, where $x_{y/\bar{z}}$ is obtained by replacing all occurrences of y with $\bar{z}$ in x.*

The replacement rule can be used to formulate an optimization procedure that replaces a child node x of a majority expression $m = \langle xyz \rangle$ with another node w if $(x \oplus w)(y \oplus z) = 0$ holds and x is not used by any other logic function in the network or as a primary output. These additional structural conditions stem from the fact that the replacement rule only enforces $x = w$ if $y \neq z$. Otherwise, if $y = z$, the result of m is determined by the majority law. However, in these cases, $x \neq w$ may hold which would affect other logic functions that use x. Further, to guarantee that the logic network stays free of cycles, the node p cannot be chosen from the transitive fan-out of m.

[2]The proof is presented in [17].

The replacement rule allows to reduce the complexity of a logic network in two ways: (i) If w is replaced by x, then x is no longer used in the logic network and can be removed. The size of the logic network is reduced if and only if x is not a constant. (ii) If the logic cone of w is smaller than x, the logic cone of m is reduced. Consequently, if multiple different nodes w satisfy the replacement rule, the x with the smallest logic cone is preferred. To find good candidate pairs x, w fast, we iterate over w in topological order, but over m in reverse topological order:

```
1  ComputeTruthTable(W);
2  foreach node m = ⟨xyz⟩ in W in reverse top. order do
3      foreach node w in W\{m} in topological order do
4          if |Fanout(x)| > 1 then continue;
5          if w ∈ TransitiveFanout(m) then continue;
6          if (x ⊕ w)(y ⊕ z) = 0 then
7              W = W[x ← w];
8              return;
9          else if (x ⊕ w̄)(y ⊕ z) = 0 then
10             W = W[x ← w̄];
11             return;
```

4 EXPERIMENTAL RESULTS

We implemented the presented size optimization methods in C++[3] and evaluated them using the EPFL combinational benchmark suite.[4] All experiments were conducted on an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz. The windows were limited to at most 12 inputs and at most 200 nodes. We apply ABC [3] equivalence checking to ensure the correct behavior of each benchmark.

4.1 MIG Size Reduction

We show the size improvement obtained by applying each of the mentioned techniques individually, and we compare our results over existing approaches from the state of the art for depth optimization [1] and size optimization [7]. In the first case, we focus on reducing the size without further increasing the depth. In the latter case, we focus on reducing the size without any additional restrictions on the depth. The proposed optimization methods achieve reductions in both cases.

Tables 1 and 2 show the results for all three optimization methods when applied individually to the benchmarks. The first column names the benchmarks, the remaining columns are organized in five blocks: the first block (**Benchmark**) lists the number of primary inputs and primary output as well as the size and depth for the benchmarks. In the second block (**Prev. flow**), we present the size and depth of the MIGs when optimized with the best-known state-of-the-art approach. The other three blocks (**FR**, **RS**, **RO**) are structured in the same way and present the size and depth after an optimization method was applied as well as the time required for optimizing the benchmark. In the last row of each table, the mean size and depth reductions are summarized for all benchmarks. The row **total reduction** shows the reductions of the benchmarks

[3]Source code for the algorithms will be published with the publication of the paper.
[4]http://lsi.epfl.ch/benchmarks

achieved by the overall synthesis flows with respect to the unoptimized benchmarks. The row **improvement** shows the reductions achieved by the new techniques with respect to the previous flow.

Starting from the depth-optimized MIGs, the three methods are capable of reducing the size of the MIGs by 2.03%, 7.12%, and 4.80%, respectively, without affecting the depth negatively. Contrarily, the depth even reduced by 0.19%, 1.01%, and 0.19%, respectively. The depth-preservation is achieved by keeping track of the depths of the nodes during logic optimization and forbidding updates on the logic network that lead to an increased depth.

Starting from the size-optimized MIGs, the three methods further reduced the size of the MIGs by 1.04%, 2.34%, and 2.83%, respectively. The size optimization also had a positive effect on the depth—the depth reduced by 2.59%, 4.97%, and 4.63%, respectively.

In these experiments, the optimization methods were applied only once. We argue that the presented techniques are, as in conventional logic synthesis, more powerful when applied several times interleaved with other optimization passes, e.g., rewriting, factoring, or balancing, which are not yet available for MIGs.

Overall, the size optimization techniques are able to regain up to 7.12% size while preserving a depth reduction of up to 44.17%. Moreover, when focusing on the size-optimized MIGs, the novel relevance optimization achieves a better size reduction than functional reduction and resubstitution, which results in an overall reduction of up to 18.13% of nodes and up to 4.63% of levels of the MIGs.

4.2 Area-Delay-Energy Product Reduction for QCA and STMG

We evaluate the efficiency of our size optimization by mapping the logic networks into QCA and STMGs. We compare our results to the state-of-the-art approach presented in [20], which we reimplemented using FR from Section 3 and windowing to achieve larger scalability and therefore address larger benchmarks. In our experiments, we found that the results obtained with our implementation of [20] outperform the more recently presented results in [6]; the latter can also easily be validated by comparing the numbers for size and depth in Table 2 from the previous section to [6, Table III] for the common benchmarks (*i2c, max, square, log2, multiplier*).

Table 3 shows area, delay, energy, and the ADE product for each of the benchmarks when mapped to QCA and STMG technologies, respectively. Total reduction compares the results to the original EPFL benchmarks, while the improvement is evaluated with respect to [20]. In this case, since STMGs and QCA technologies have limited possibilities for inverter implementation, we always applied the algorithm presented in [19] in order to create inversion free circuits. For size optimization we used the approach in [7] followed by the FR, RO, and RS optimization techniques. To obtain area, delay, and energy, we use the same specifications as in [19]. Our optimization method is able to further reduce the ADE product by 2.31% for QCA and by 2.07% for STMGs. Overall, the MIG-based synthesis flow is able to obtain an average improvement of 20.81% and 55.63% for QCA and STMG, respectively.

5 CONCLUSIONS

Majority-based logic synthesis is a key enabler for majority-based nano-emerging technologies. In this paper, we stocked up the repertoire of optimization techniques for MIGs with powerful size optimization methods in order to exploit the full potential of MIGs for majority-based nano-emerging technologies. Experimental results for the proposed optimization methods confirmed the effectiveness of our proposed optimization techniques. They show significant size reductions when compared to the state-of-the-art depth and size optimizations for MIGs. We also show the gain after mapping the optimized logic networks to the technologies. Overall logic minimization accounts for a total reduction of 20.81% and 55.63% of the area-delay-energy products for STMG and QCA, respectively.

ACKNOWLEDGMENTS

This research was supported by H2020-ERC-2014-ADG 669354 CyberCare and the Swiss National Science Foundation (200021-169084 MAJesty and 200021-146600).

REFERENCES

[1] Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2014. Majority-Inverter Graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization. In *Design Automation Conference*. 194:1–194:6.
[2] Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2016. Majority-Inverter Graph: A New Paradigm for Logic Optimization. *IEEE Trans. on CAD of Integrated Circuits and Systems* 35, 5 (2016), 806–819.
[3] Robert K. Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification*. 24–40.
[4] Randal E. Bryant. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers* 35, 8 (1986), 677–691.
[5] Yung-Chih Chen and Chun-Yao Wang. 2009. Fast detection of node mergers using logic implications. In *2009 International Conference on Computer-Aided Design, ICCAD 2009, San Jose, CA, USA, November 2-5, 2009*. 785–788.
[6] Chun-Che Chung, Yung-Chih Chen, Chun-Yao Wang, and Chia-Cheng Wu. 2017. Majority logic circuits optimisation by node merging. In *Asia and South Pacific Design Automation Conference*. 714–719.
[7] Winston Haaswijk, Mathias Soeken, Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2017. A novel basis for logic optimization. In *Asia and South Pacific Design Automation Conference*. 151–156.
[8] Alexander Khitun and Kang L. Wang. 2011. Non-volatile magnonic logic circuits engineering. *Journal of Applied Physics* 110, 034306 (2011).
[9] Kun Kong, Yun Shang, and Ruqian Lu. 2010. An optimized majority logic synthesis methodology for quantum-dot cellular automata. *IEEE Trans. on Nanotechnology* 9, 2 (2010), 170–183.
[10] Craig S. Lent, P. Doublas Tougaw, Wolfgang Porod, and Gary H. Bernstein. 1993. Quantum cellular automata. *Nanotechnology* 4, 1 (1993), 49–57.
[11] Alan Mishchenko and Robert K. Brayton. 2006. Scalable logic synthesis using a simple circuit structure. In *Int'l Workshop on Logic and Synthesis*. 15–22.
[12] Dmitri E. Nikonov, George I. Bourianoff, and Tahir Ghani. 2011. Nanomagnetic circuits with spin torque majority gates. In *Int'l Conf. on Nanotechnology*. 1384–1388.
[13] Dmitri E. Nikonov, George I. Bourianoff, and Tahir Ghani. 2011. Proposal of a Spin Torque Majority Gate Logic. *IEEE Electron Device Letters* 32, 8 (2011), 1128–1130.
[14] Sayak Ray, Alan Mishchenko, Niklas Eén, Robert K. Brayton, Stephen Jang, and Chao Chen. 2012. Mapping into LUT structures. In *Design, Automation and Test in Europe*. 1579–1584.
[15] Tsutomu Sasao. 1999. *Switching Theory for Logic Synthesis*. Springer.
[16] Mathias Soeken, Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2016. Optimizing Majority-Inverter Graphs with functional hashing. In *Design, Automation and Test in Europe*. 1030–1035.
[17] Eleonora Testa, Mathias Soeken, Luca Amaru, Winston Haaswijk, and Giovanni De Micheli. 2017. Mapping Monotone Boolean Functions Into Majority. *Submitted* (2017).
[18] Eleonora Testa, Mathias Soeken, Odysseas Zografos, Luca Gaetano Amarù, Praveen Raghavan, Rudy Lauwereins, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2016. Inversion optimization in Majority-Inverter Graphs. In *Int'l Symp. on Nanoscale Architectures*. 15–20.
[19] Eleonora Testa, Odysseas Zografos, Mathias Soeken, Adrien Vaysset, Mauricio Manfrini, Rudy Lauwereins, and Giovanni De Micheli. 2017. Inverter Propagation and Fan-Out Constraints for Beyond-CMOS Majority-Based Technologies. In *Annual Symp. on VLSI*. 164–169.
[20] Peng Wang, Mohammed Y. Niamat, Srinivasa R. Vemuru, Mansoor Alam, and Taylor Killian. 2015. Synthesis of majority/minority logic networks. *IEEE Trans. on Nanotechnology* 14, 3 (2015), 473–483.

Table 1: Optimization Methods Applied to Depth Optimized Benchmarks

Benchmark				Prev. flow [1]		FR [6]			RS			RO		
	I/O	Size	Depth	Size	Depth	Size	Depth	Time [s]	Size	Depth	Time [s]	Size	Depth	Time [s]
dec	8 / 256	304	3	304	3	304	3	0.00	304	3	0.14	304	3	0.00
ctrl	7 / 26	174	10	180	6	176	6	0.03	169	6	0.09	160	6	0.05
bar	135 / 128	3336	12	3336	12	3336	12	2.07	3199	12	12.77	3336	12	3.27
cavlc	10 / 11	693	16	708	11	701	11	0.20	695	11	0.56	680	11	0.24
int2float	11 / 7	260	16	260	8	260	8	0.06	255	8	0.17	244	8	0.06
i2c	147 / 142	1342	20	1457	9	1454	9	0.37	1423	9	1.24	1420	9	0.49
router	60 / 30	257	54	365	26	362	26	0.16	349	26	1.22	363	26	0.21
voter	1001 / 1	13758	70	14075	60	12838	58	63.48	11157	58	224.40	12670	57	66.82
arbiter	256 / 129	11839	87	11523	77	11523	77	13.22	11523	77	30.44	11523	77	14.67
mem_ctrl	1204 / 1231	46836	114	52103	79	51816	79	361.77	49889	77	1818.65	51524	80	484.01
sin	24 / 25	5416	225	6679	112	6523	111	9.08	6138	108	64.08	6409	112	13.28
square	64 / 128	18484	250	21927	39	21548	39	79.12	19253	38	579.54	21081	39	124.04
priority	128 / 8	978	250	1148	126	1148	126	0.65	1050	126	7.10	951	126	1.10
adder	256 / 129	1020	255	1859	18	1760	18	1.05	1666	18	4.68	1825	18	0.98
multiplier	128 / 128	27062	274	33355	113	32952	112	178.21	31425	106	1670.05	30790	113	563.70
max	512 / 130	2865	287	4754	48	4666	48	5.48	4658	48	17.69	4677	48	6.38
log2	32 / 32	32060	444	36936	262	36406	262	209.01	34233	223	1660.81	35843	262	324.71
sqrt	128 / 64	24618	5058	32138	3430	31808	3447	137.56	30429	3430	611.02	31842	3430	154.10
total reduction				-19.09%	+43.16%	-17.06%	+43.35%		-11.97%	+44.17%		-14.28%	+43.35%	
improvement				0.00%	0.00%	+2.03%	+0.19%		+7.12%	+1.01%		+4.80%	+0.19%	

Table 2: Optimization Methods Applied to Size Optimized Benchmarks

Benchmark				Prev. flow [7]		FR [6]			RS			RO		
	I/O	Size	Depth	Size	Depth	Size	Depth	Time [s]	Size	Depth	Time [s]	Size	Depth	Time [s]
ctrl	7 / 26	174	10	139	10	139	10	0.01	128	9	0.06	135	9	0.02
router	60 / 30	257	54	220	54	217	54	0.08	215	54	1.05	211	54	0.11
int2float	11 / 7	260	16	263	18	261	16	0.04	256	16	0.46	254	16	0.06
dec	8 / 256	304	3	328	4	328	4	0.00	328	4	0.26	328	4	0.01
cavlc	10 / 11	693	16	757	19	744	19	0.29	725	19	2.43	724	19	0.41
priority	128 / 8	978	250	993	245	993	245	2.15	978	239	9.95	807	125	0.92
adder	256 / 129	1020	255	386	129	386	129	0.08	385	129	1.29	386	129	0.07
i2c	147 / 142	1342	20	1329	23	1310	23	0.42	1298	23	2.78	1287	23	0.64
max	512 / 130	2865	287	2491	290	2428	261	4.72	2469	280	18.52	2448	279	5.70
bar	135 / 128	3336	12	3110	14	3110	14	2.40	3110	13	14.76	3088	14	3.75
sin	24 / 25	5416	225	4496	167	4480	162	4.74	4465	158	36.31	4480	170	6.08
arbiter	256 / 129	11839	87	8957	63	8957	63	8.82	8957	63	45.41	8957	63	11.97
voter	1001 / 1	13758	70	7767	67	6649	59	31.90	5787	47	87.81	6537	61	45.10
square	64 / 128	18484	250	13671	156	13390	130	61.67	13194	128	109.84	13463	154	47.55
sqrt	128 / 64	24618	5058	21066	5989	21063	5989	102.62	20976	5942	624.11	21060	5988	109.43
multiplier	128 / 128	27062	274	19844	143	19824	143	72.16	19824	141	252.05	19804	143	122.00
log2	32 / 32	32060	444	25040	230	24999	230	89.96	24996	229	257.55	24977	230	109.43
mem_ctrl	1204 / 1231	46836	114	45034	144	44476	144	410.72	43305	136	1170.23	44118	143	600.50
total reduction				+15.30%	+5.59%	+16.34%	+8.18%		+17.64%	+10.56%		+18.13%	+10.22%	
improvement				0.00%	0.00%	+1.04%	+2.59%		+2.34%	+4.97%		+2.83%	+4.63%	

Table 3: Size Optimization Techniques (after QCA and STMG Technology Mapping)

Benchmark	Baseline [20] QCA				Opt. QCA				Baseline [20] STMG				Opt. STMG			
	Area [μm^2]	Delay [ns]	Energy [J]	ADE	Area [μm^2]	Delay [ns]	Energy [J]	ADE	Area [μm^2]	Delay [ns]	Energy [J]	ADE	Area [μm^2]	Delay [ns]	Energy [J]	ADE
adder	1.6	0.5	4.0E-18	3.5E-18	1.6	0.5	4.0E-18	3.5E-18	15.4	193.5	5.4E-11	1.6E-07	15.4	193.5	5.4E-11	1.6E-07
arbiter	12.8	0.3	3.1E-17	1.1E-16	12.8	0.3	3.1E-17	1.1E-16	35.2	94.5	5.9E-10	2.0E-06	35.2	94.5	5.9E-10	2.0E-06
bar	7.9	0.1	1.9E-17	1.1E-17	7.8	0.1	1.9E-17	1.0E-17	21.9	21.0	7.2E-10	3.3E-07	21.9	19.5	7.1E-10	3.0E-07
cavlc	1.4	0.1	3.3E-18	4.1E-19	1.3	0.1	3.2E-18	3.7E-19	4.0	28.5	1.3E-10	1.5E-08	3.8	28.5	1.2E-10	1.3E-08
ctrl	0.3	0.1	6.2E-19	8.6E-21	0.2	0.1	5.7E-19	6.6E-21	0.7	15.0	2.3E-11	2.4E-10	0.6	13.5	2.1E-11	1.8E-10
dec	0.4	0.0	1.1E-18	1.4E-20	0.4	0.0	1.1E-18	1.4E-20	1.2	6.0	2.8E-11	2.0E-10	1.2	6.0	2.8E-11	2.0E-10
i2c	2.6	0.1	6.5E-18	1.8E-18	2.5	0.1	6.1E-18	1.6E-18	6.9	34.5	2.1E-10	5.0E-08	6.6	34.5	2.0E-10	4.6E-08
int2float	0.5	0.1	1.2E-18	5.0E-20	0.5	0.1	1.2E-18	4.3E-20	1.4	24.0	4.8E-11	1.6E-09	1.3	24.0	4.4E-11	1.4E-09
log2	60.0	0.9	1.5E-16	8.2E-15	59.9	0.9	1.5E-16	8.2E-15	179.6	345.0	2.1E-09	1.3E-04	179.3	343.5	2.0E-09	1.2E-04
max	7.4	1.1	1.8E-17	1.4E-16	7.3	1.1	1.8E-17	1.4E-16	30.7	391.5	4.3E-10	5.2E-06	30.7	390.0	4.3E-10	5.1E-06
mem	92.4	0.6	2.3E-16	1.2E-14	86.3	0.6	2.1E-16	1.0E-14	265.2	216.0	6.5E-09	3.7E-04	247.0	204.0	6.0E-09	3.0E-04
mult	47.7	0.6	1.2E-16	3.3E-15	47.7	0.6	1.2E-16	3.2E-15	141.6	214.5	2.2E-09	6.6E-05	141.6	211.5	2.2E-09	6.5E-05
priority	2.6	1.0	6.5E-18	1.7E-17	2.6	1.0	6.3E-18	1.6E-17	7.7	367.5	1.8E-10	5.0E-07	7.7	358.5	1.7E-10	4.8E-07
router	0.7	0.2	1.8E-18	3.1E-19	0.7	0.2	1.8E-18	2.9E-19	3.6	81.0	4.2E-11	1.2E-08	3.6	81.0	4.1E-11	1.2E-08
sin	10.7	0.7	2.6E-17	1.9E-16	10.6	0.7	2.6E-17	1.8E-16	31.8	243.0	3.5E-10	2.7E-06	31.6	240.0	3.3E-10	2.5E-06
sqrt	51.0	24.0	1.3E-16	1.5E-13	50.8	23.8	1.2E-16	1.5E-13	151.6	8983.5	1.6E-09	2.2E-03	150.9	8911.5	1.6E-09	2.2E-03
square	32.0	0.5	7.8E-17	1.3E-15	31.0	0.5	7.6E-17	1.3E-15	95.1	195.0	1.6E-09	3.0E-05	92.3	193.5	1.6E-09	2.8E-05
voter	19.9	0.3	4.9E-17	2.4E-16	17.6	0.2	4.3E-17	1.6E-16	60.1	88.5	6.7E-10	3.6E-06	60.1	73.5	5.8E-10	2.6E-06
total reduction				+18.50%				+20.81%				+53.56%				+55.63%
improvement				0.00%				+2.31%				0.00%				+2.07%