Late Breaking Results: Majority-Inverter Graph Minimization by Design Space Exploration

Siang-Yun Lee
EPFL, Switzerland

Alessandro Tempia Calvino
EPFL, Switzerland

Heinz Riener
Cadence Design Systems, Germany

Giovanni De Micheli
EPFL, Switzerland

Abstract

The majority-inverter graph (MIG) is a homogeneous logic network widely used in logic synthesis for majority-based emerging technologies. Many logic optimization algorithms have been proposed for MIGs, including rewriting, resubstitution, and graph mapping. However, unlike AIGs, research on optimization flows for MIGs is limited. In this paper, we explore combinations of well-developed MIG optimization algorithms using an on-the-fly design space exploration strategy and present the latest best results on MIG size minimization of EPFL benchmarks. Significant reductions (of **88%** and **79%**) are observed for two specific benchmarks and an average of **14%** improvement is achieved compared to the state-of-the-art flow.

1 Introduction

The *Majority-Inverter Graph* (MIG) is a directed acyclic graph modeling combinational circuits, where each node represents a 3-input *majority* (MAJ) gate and edges represent wires with optional inverters [2]. A MAJ gate computes the *majority-of-three* function of its inputs: $M(a,b,c) = ab + bc + ac$. The size of a MIG is measured by its number of gates, and the depth of a MIG is the length of the longest path from a primary input to a primary output. MIGs are a generalization of *And-Inverter Graphs* (AIGs) because an AIG can be directly converted into a MIG by translating each AND gate as a MAJ gate with a constant-0 input. Being a more compact representation compared to AIGs, MIGs have an advantage in delay optimization, especially for arithmetic-rich circuits, and have contributed to improvements in standard cell and FPGA design flows [2]. Moreover, MIGs are also heavily used in logic synthesis for majority-based emerging technologies [6, 12], such as the superconductive *Adiabatic Quantum-Flux Parametron* (AQFP) circuits [9], *Spin-Torch Majority Gate* (STMG) logic [5], and *Quantum Cellular Automata* (QCA) [13].

Modern scalable logic synthesis algorithms tend to leverage simple and homogeneous technology-independent network data structures like AIGs for their efficiency. Most AIG-based optimization methodologies can be adapted for MIGs, and tailored algorithms specialized for MIG optimization have also been researched. To name a few, algebraic rewriting applies MAJ-specific algebraic rules to optimize MIGs [2]; cut rewriting, resubstitution, and refactoring are Boolean optimization algorithms originally developed for AIGs which have been adapted for MIGs [6]; graph mapping can be applied from MIGs to MIGs (remapping) for optimization purpose [11].

Recent works on MIG optimization often adopt flows consisting of existing and newly proposed algorithms to demonstrate their optimization capability. In [12], a flow iterating algebraic rewriting, resubstitution, and refactoring until convergence is used to optimize MIGs for AQFP technology mapping. In [4], a MIG flow consisting of AIG-based pre-optimization, graph mapping, lightweight resubstitution, and high-effort resubstitution is presented to demonstrate the strength of the proposed algorithm. Finally, the latest work [10] adds don't-care-based rewriting and remapping on top of the flow in [4]. However, these works present somewhat ad-hoc flows mainly for the purpose of showcasing the proposed algorithms. It is unclear what the best MIG optimization flow should be like. While improvements have been made from one work to the next, the limits are seldom explored. Thus, in this work, we aim to lower the upper bounds on the problem of MIG size optimization.

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3663507

Algorithm 1: On-the-fly design space exploration

```
Input: Original MIG $N_0$
Output: Best-seen MIG $N_{\text{best}}$ with the smallest size
1 foreach restart = 1 upto num_restarts do
2     $N_{\text{curr}} \leftarrow N_0$
3     foreach step while not timeout and not bailout do
4         decompress($N_{\text{curr}}$)
5         compress($N_{\text{curr}}$)
6         if $|N_{curr}| < |N_{best\_inner}|$ then $N_{\text{best\_inner}} \leftarrow N_{\text{curr}}$
7     if $|N_{best\_inner}| < |N_{best}|$ then $N_{\text{best}} \leftarrow N_{\text{best\_inner}}$
8 return $N_{best}$
```

2 Methods

We perform on-the-fly design space exploration to search for better local optima in terms of MIG size. Minimizing MIG size helps reduce circuit area for most technologies. For superconductive circuits, reducing gate count also reduces the number of active components (Josephson junctions) and thus decreases switching power consumption.

The exploration process is outlined in Algorithm 1. The algorithm has an outer loop (lines 1-7), where each iteration is called a *restart*, and an inner loop (lines 3-6), where each iteration is called a *step*. The intuition is to apply a long sequence of logic synthesis algorithms on the original network N_0, continue without rolling back ("undoing") even if the size of the network becomes bigger in the process, and record the best result seen in each restart (N_{best_inner}) as well as among all restarts (N_{best}). The inner loop breaks on either a *timeout* condition, when the execution time of the current restart exceeds a predefined length, or a *bailout* condition, when N_{best_inner} have not been updated for more than a certain number of steps. The outer loop serves as an occasional reset to the starting point such that different trajectories in the design space may be explored.

The main optimization process happens in the `decompress` and `compress` functions (lines 4-5). In `decompress`, a script is randomly chosen from a predefined set of decompressing scripts and applied, aiming at restructuring the network drastically and possibly increasing its size. Applying decompressing scripts helps escape from local minima and achieve the "hill climbing" effect in the non-convex design space. In our experiment, we use the following set of decompressing scripts:

(1) Convert the MIG into an AIG by replacing each MAJ gate with four AND gates using $M(a, b, c) = ab + c(a + b)$; apply ABC[1] [3] commands `&dch` (choice computation), `&if` (k-LUT mapping, with $2 \leq k \leq 6$), `&mfs` (resubstitution on the k-LUT network), and `&st` (convert to AIG); convert the resulting AIG back into MIG with versatile graph mapping [11].
(2) Perform LUT mapping on the MIG to obtain a k-LUT network, where $3 \leq k \leq 6$; convert the k-LUT network back into MIG either by *disjoint support decomposition* (DSD) and Shannon decomposition or by *sum-of-product* (SOP) factoring.
(3) Decompose each MAJ gate without constant input into four MAJ gates with constant inputs using the formula in (1).

Similarly, the function `compress` randomly applies a script chosen from a predefined set of compressing scripts, which aims at optimizing the MIG and reducing its size as much as possible. In our experiment, we have used some subsets of the following set:

(a) Convert the MIG into an AIG as in decompressing script (1); apply ABC script `resyn2rs` (AIG optimization flow); convert the resulting AIG back into MIG with graph mapping.
(b) Apply graph remapping [11] with one round of area flow and two rounds of exact area optimization, optionally with logic sharing and/or satisfiability don't cares enabled.
(c) Apply MIG rewriting [10] with a cut size of 4, optionally with satisfiability don't cares computed using a window size of 8.
(d) Apply simulation-guided MIG resubstitution [4] with a window size of 6 or 8 and maximum dependency circuit size between 0 and 7.
(e) Apply three times remapping (as in (b)); apply three times rewriting (as in (c)); apply once resubstitution (as in (d)).

3 Experimental Results

We report the latest best results for MIG size minimization on the EPFL benchmark suite [1] in Table 1. The EPFL benchmark suite consists of un-optimized arithmetic and random control benchmarks commonly used to test the capabilities of logic synthesis tools. The results are obtained using the logic synthesis library *mockturtle*[2] [7, 8] and are made available online[3]. The number of restarts is 4, the bailout condition is 50 steps without improvement, and the timeout condition is $\max\{|N_0|/10, 1000\}$ seconds per restart.

In Table 1, the MIG size and depth are reported, as well as the improvement in size compared to the state-of-the-art flow (SoTA) consisting of AIG-based pre-optimization, remapping with don't cares, rewriting with don't cares, and resubstitution [10]. The depths of our results are reported here for completeness, but it is not the optimization objective in this paper (and was not reported in [10]).

We have tried three settings of the compressing script set: {(a), (b) without don't cares, (d)}, {(a), (b), (c), (d)}, and {(a), (e)}. Moreover, we also tried to use the results in [10] as the starting point, in addition to starting from the original MIGs directly converted from AIGs. Results presented in Table 1 are the best ones we obtained and are not all from the same run. Nevertheless, the total sizes of each run are within 1% difference to the total of the collective best, indicating a consistent improvement.

For the benchmarks "adder" and "dec", we observe that almost all methods and flows, whether simple or complicated, obtain exactly the same results. It is thus very likely that these results are the global optimum and can never be further improved. In contrast, for the benchmarks "arbiter" and "mem_ctrl", we observe drastic improvements of 88.2% and 78.5%, respectively, achieved by our design space exploration compared to any other existing flows.

[1]Available: https://github.com/berkeley-abc/abc
[2]Available: https://github.com/lsils/mockturtle
[3]Available: https://github.com/lsils/benchmarks-mig/

Table 1. Latest best results for MIG size optimization.

	SoTA [10]	New Best Results		
Bench.	Size	Size	Impr.	Depth
adder	384	384	-	129
bar	2433	1906	21.7%	15
div	12462	12368	0.8%	2251
hyp	115541	115539	0.002%	9129
log2	22010	22008	0.01%	184
max	2190	1939	11.5%	172
multiplier	17112	17112	-	137
sin	3870	3869	0.03%	124
sqrt	12357	12247	0.9%	2156
square	8138	8089	0.6%	126
arbiter	6711	792	**88.2%**	108
cavlc	492	374	24.0%	16
ctrl	74	60	18.9%	8
dec	304	304	-	3
i2c	871	636	27.0%	16
int2float	172	115	33.1%	9
mem_ctrl	32097	6886	**78.5%**	26
priority	406	337	17.0%	23
router	147	97	34.0%	13
voter	4555	3894	14.5%	32
Total	242326	208956	13.8%	14677

Overall, a 13.8% improvement is achieved compared to the state of the art.

4 Conclusions

In this late-breaking-results paper, we present our latest best results on MIG size optimization and provide some insights into what a good MIG flow would be like. As the Boolean optimization problem faced in logic synthesis is intractable and global optimum results are typically unknown, this work contributes to finding better upper bounds on the minimal size of common benchmarks, which in turn helps researchers evaluate new algorithms in the future. In future work, we plan to run a similar study on MIG depth minimization, which is important for latency optimization in many technologies.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Proceedings of IWLS*.
[2] Luca Amaru, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. Majority-inverter graph: A new paradigm for logic optimization. *IEEE Trans. on CAD* 35, 5 (2015), 806–819.
[3] Robert K. Brayton and Alan Mishchenko. 2010. ABC: An Academic Industrial-Strength Verification Tool. In *Proceedings of CAV*. 24–40.
[4] Siang-Yun Lee and Giovanni De Micheli. 2023. Heuristic Logic Resynthesis Algorithms at the Core of Peephole Optimization. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* (2023).
[5] Dmitri E Nikonov, George I Bourianoff, and Tahir Ghani. 2011. Proposal of a spin torque majority gate logic. *IEEE Electron Device Letters* (2011).
[6] Heinz Riener, Eleonora Testa, Luca G. Amarù, Mathias Soeken, and Giovanni De Micheli. 2018. Size Optimization of MIGs with an Application to QCA and STMG Technologies. In *Proceedings of NANOARCH*. 157–162.
[7] Heinz Riener, Eleonora Testa, Winston Haaswijk, Alan Mishchenko, Luca G. Amarù, Giovanni De Micheli, and Mathias Soeken. 2019. Scalable Generic Logic Synthesis: One Approach to Rule Them All. In *Proceedings of DAC*. 70.
[8] Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, Siang-Yun Lee, Alessandro Tempia Calvino, Dewmini Sudara Marakkalage, and Giovanni De Micheli. 2022. The EPFL Logic Synthesis Libraries. arXiv:1805.05121 http://arxiv.org/abs/1805.05121
[9] Naoki Takeuchi, Dan Ozawa, Yuki Yamanashi, and Nobuyuki Yoshikawa. 2013. An adiabatic quantum flux parametron as an ultra-low-power logic device. *Superconductor Science and Technology* 26, 3 (2013), 035010.
[10] Alessandro Tempia Calvino and Giovanni De Micheli. 2024. Scalable Logic Rewriting Using Don't Cares. In *Proceedings of DATE*.
[11] Alessandro Tempia Calvino, Heinz Riener, Shubham Rai, Akash Kumar, and Giovanni De Micheli. 2022. A Versatile Mapping Approach for Technology Mapping and Graph Optimization. In *Proceedings of ASP-DAC*. 410–416.
[12] Eleonora Testa, Siang-Yun Lee, Heinz Riener, and Giovanni De Micheli. 2021. Algebraic and Boolean Optimization Methods for AQFP Superconducting Circuits. In *Proceedings of ASP-DAC*. 779–785.
[13] P Douglas Tougaw and Craig S Lent. 1994. Logical devices implemented using quantum cellular automata. *Journal of Applied physics* 75, 3 (1994), 1818–1825.