INVERSIONS OPTIMIZATION IN XOR-MAJORITY GRAPHS WITH AN APPLICATION TO QCA

Lei Shi, and Zhufei Chu[*]
EECS, Ningbo University, Ningbo 315211, China
* Email: chuzhufei@nbu.edu.cn

ABSTRACT

Inversions are indispensable to build a logically complete Boolean system. However, the implementations of inversion in some nanotechnologies are expensive than the other logical operations. Therefore, the inversions optimization is of paramount interest for high-performance nanotechnology circuit design. Recently, XOR-Majority Graphs (XMGs) are used as logic representations for advanced logic synthesis. To this end, we propose an XMG optimization technique to rewrite the complemented edges while not changing its shape. The optimizations consider both majority-of-three (MAJ) nodes and exclusive-OR (XOR) nodes by using inverter propagations. The experimental results on EPFL benchmark suites show our method can achieve an average reduction of 17.3% number of inversions, which brings up to 9.8% area improvement for the implementation using Quantum-dot Cellular Automata (QCA) circuits.

INTRODUCTION

Nanotechnologies have emerged as a promising replacement for complementary-metal-oxide-semiconductor (CMOS) technique. Many of the nanotechnologies realize majority-of-three (MAJ) function as a building primitive instead of NAND/NOR used in CMOS. Quantum-dot Cellular Automata (QCA) is one of such nanotechnologies [1], which enables efficient circuit design with high device density, high switching speed, and low power consumption. However, the implementation cost for inversion is expensive than the other logical operations in QCA [2]. For example, the layouts of the MAJ, exclusive-OR (XOR), and inverter implemented by QCA are shown in Figure 1 (a), (b), and (c), respectively. In terms of the number of QCA cells, 13 cells are required for an inverter while 5 for a MAJ and 12 for an XOR. Therefore, it is important to carry out inversion optimization in logic level for high-performance QCA circuit design.

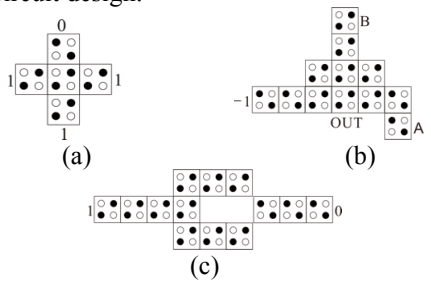

Figure 1: QCA layout for (a) MAJ, (b) XOR, and (c) inverter

The logic representation is important for technical physical implementation. In past decades, there is a shift from complicated heterogeneous logic representations to simpler homogeneous ones, such as and-inverter graphs (AIGs) [3] and majority-inverter graphs (MIGs) [4]. The uniform data structures make logical representation and optimization more efficient. This is because they are easy to manipulate and using fewer memories. For the sake of more compact representations in exact synthesis, XOR-majority graphs (XMGs) are extended by MIGs in the way of additionally introduce XOR operations. XMGs have been applied to various applications, such as quantum circuit synthesis, exact-synthesis-aware rewriting [5], and QCA circuit design [6].

In this paper, we present a heuristic algorithm that targets at optimizing the number of inversions in XMGs while not changing its shapes. By one-to-one mapping of the optimized XMGs to QCA circuits, we can achieve area improvement after optimization. In the remainder of the paper, we first address the inverter propagation rules of MAJ and XOR operations. Then the inversion optimization algorithm is demonstrated. The experimental results over EPFL benchmark suites show the number of inverters has an averagely 17.3% reduction, which in turn brings up to 9.8% area improvement for QCA circuits.

BACKGROUND

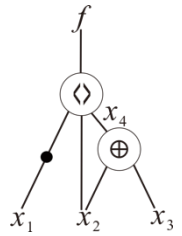

Figure 2: An XMG logic network

The MAJ function can be represented in disjunctive, conjunctive normal forms, and exclusive-sum-of-products (ESOP) form as

$$\begin{aligned}\langle xyz\rangle &= xy + xz + yz = (x+y)(x+z)(y+z)\\ &= xy \oplus xz \oplus yz \end{aligned} \quad (1)$$

where '⟨ ⟩' is the MAJ operation. By setting one input of MAJ to constants, the traditional AND and OR operators can be also obtained, respectively.

$$\langle 0xy\rangle = xy \text{ and } \langle 1xy\rangle = x + y$$

An XMG is a logic network in which each node

corresponds to either a MAJ or an XOR operator. The edges between nodes can be complemented to indicate the insertion of an inverter. Figure 2 shows an example Boolean logic network represented by an XMG, in which $x_4 = x_2 \oplus x_3$ and $f = \langle \overline{x_1} x_2 x_4 \rangle$.

INVERSION OPTIMIZATION

Inversion Propagation Rules

The transformation rules do not change the depth nor the size of XMGs. Different rules would be adopted depending on how many complemented edges exist in the node inputs and the type of the nodes, e.g., MAJ and XOR nodes should use different optimization rules.

Given a MAJ node, considering the optimization situation, it generally can be classified into three cases:

I $\quad \langle \bar{x}\bar{y}\bar{z} \rangle = \overline{\langle xyz \rangle}$

II $\quad \overline{\langle \bar{x}\bar{y}z \rangle} = \langle xy\bar{z} \rangle$

III $\quad \langle x\bar{y}\bar{z} \rangle = \overline{\langle \bar{x}yz \rangle}$

In terms of the XOR node, the inverters can be propagated like the way in MAJ's inverter propagation.

IV $\quad \bar{x} \oplus \bar{y} = x \oplus y$

V $\quad \bar{x} \oplus y = x \oplus \bar{y} = \overline{x \oplus y}$

Take Figure 3(a) as an example, we consider each node in the XMG, for the node *a*, it is a MAJ node with three complemented inputs, by applying the rule I for optimization, the number of inverters is reduced from 3 to 1; also, the node *b* satisfies the rule II. After optimizing these two MAJ nodes, the optimized XMG is shown in Figure 3(b). At last, we consider the XOR node *c*, it has two complemented inputs and fulfills the rule IV, we finally get the optimized XMG shown in Figure 3(c).

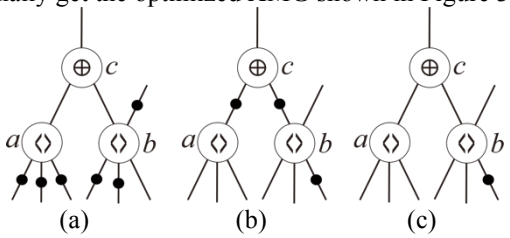

Figure 3: One-level optimization in XMGs, (a) initial XMG, (b) after inversion optimization of nodes a and b, (c) final XMG after optimizing node c.

Note that rule III may not reduce the number of inverters, but it opens an opportunity for further optimization due to the different inversion configuration. For instance, an XMG with three MAJ nodes is shown in Figure 4(a). According to propagation rules, none of these three nodes can be applied using the optimization rules. However, the node *a* can execute the rule III, though the total number of complemented edges remains. By applying a transformation on the MAJ node *a* with the result in Figure 4(b), the node *c* satisfies the rule II. Finally, we obtained the optimized XMG shown in Figure 4(c). The number of inverters is reduced from 4 to 3.

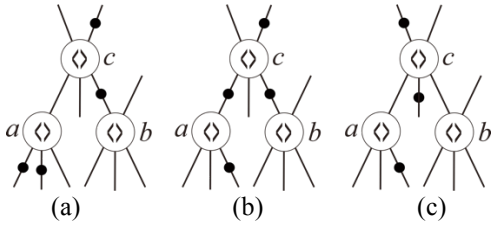

Figure 4: Two-level optimization in XMGs

The rule V indicates the inversion of the XOR node can be configured to any other input or output. Take Figure 5(a) as an example, the XOR node *c* has one complemented input on the right child. However, due to the rule V, the inversion can be moved to the left child, which is the output of node *a*, thus the optimization opportunity arises for node *a*, to finally get the optimized XMG in Figure 5(b).

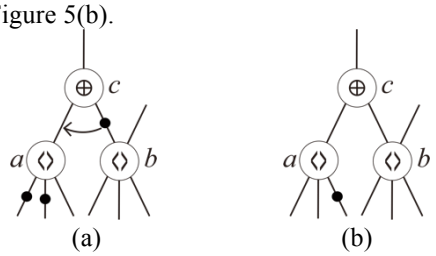

Figure 5: (a) Inversion configuration of the XOR node, (b) the optimized XMG.

One-level optimization

The optimization is implemented for each node in an XMG in topological order, which is referred to as one-level optimization as we only consider the inversions of one node. The formula of computing potential savings of inverters is listed as follows:

$$Savings = \#COM - \#NONCOM \qquad (2)$$

where *#COM* is the total number of complemented edges for a node in both its inputs and outputs, while *#NONCOM* is the number of normal edges.

Considering a MAJ node or an XOR node as the target node, we estimate each node's *Savings* if (2) is larger or equal than 0, which is used as the cost function during optimization. In details, if *Savings* is larger or equal than 0, the node executes one of the transformation rules from I to V. Even though the number of inverters remains when *Savings* is equal to 0, the inverter configurations are changed substantially.

Two-level optimization

As shown in Figure 4, the number of inverters on node *a* does not change but affects the inverter distribution of its parent node *c* and produces optimization possibilities. Therefore, as [2] did, we also perform two-level

optimization that evaluates each node with its parents. First, we regard the node whose one-level *Savings* is equal or less than 0 as the target node. Then we compute the *Savings* of the target node and its parent nodes. If the total *Savings* of the target node and its parent nodes are larger than 0, the propagation rules can be applied to the target node and its parent nodes. This transformation changes the complemented edges pattern of the parent nodes, thus providing more optimization opportunities.

Optimization Algorithm

Inversion optimization:

Input: an original XMG
Output: an optimized XMG
1 `one_level_opt()`
2 `two_level_opt()` //initialization
3 **while** the number of inverter is decreasing **do**
4 `one_level_opt()`
5 `two_level_opt()`
6 **end**

The proposed inversion optimization algorithm is shown above. We first apply both one-level and two-level optimization, if any improvement is achieved, the algorithm is aggressively applied to the XMGs until no more improvement can be obtained.

EXPERIMENTAL RESULTS

We developed a C++ program to implement the proposed inversion optimization algorithm on an Intel Xeon CPU at 2.20GHz with 15.6GB of main memory. We use "`xmglut -k 4`" command in the open source logic synthesis tool CirKit[1] to generate the original XMGs. The optimized netlists are verified by ABC[2] for functional correctness.

Our experimental results over EPFL benchmark suites are shown in TABLE I, where "Benchmark" lists the benchmark name, primary inputs/outputs (PIs/POs), size and depth information. "QCI" indicates the area improvement using QCA circuit after optimization. The results (which list the number of inverters in the original and optimized XMGs) indicate the proposed inversion optimization algorithm in XMGs is able to optimize the number of complemented edges by 17.3% on average. To evaluate the impact on QCA circuits, we consider that 9 cells are used to build each wire, and 4 more cells are necessary on each wire to invert the signal. Consequently, the number of QCA cells can be reduced up to 9.8% while 2.1% on average.

TABLE I. EXPERIMENTAL RESULTS OF THE BENCHMARKS

Benchmark	PI/PO	Size	Depth	#INV_origin [5]	#INV_opt	Improv.	QCI
log2	32/32	32060	444	14859	8624	42.0%	3.7%
max	512/130	2865	287	1253	1167	6.9%	0.5%
ctrl	7/26	174	10	66	58	12.1%	0.9%
sin	24/25	5416	225	2520	2281	9.5%	0.8%
i2c	147/142	1342	20	1113	1050	5.7%	0.6%
div	128/128	57247	4372	29104	20116	30.9%	9.8%
arbiter	256/129	11839	87	10111	8348	17.4%	1.8%
hyp	256/128	214335	24801	43773	33448	23.6%	1.1%
mem_ctrl	1204/1231	46836	114	31151	28594	8.2%	0.8%
multiplier	128/128	27062	274	7397	6201	16.2%	1.0%
Average						17.3%	2.1%

SUMMARY

In this paper, we proposed an inversion optimization algorithm in XMGs. By exploiting the propagation rules of both MAJ and XOR nodes, the optimization results over EPFL benchmark suites indicate we can achieve 17.3% reduction on the number of inversions, which in turn has a positive impact on QCA circuit realization.

ACKNOWLEDGEMENTS

This research was partly supported by NSF China 61501268 and 61871242.

[1] https://github.com/msoeken/cirkit
[2] https://github.com/berkeley-abc/abc

REFERENCES

[1] C. S. Lent, P. D. Tougaw, W. Porod and G. H. Bernstein. *Nanotechnology*, Vol. 4, No. 1, 1993, pp. 49-57.
[2] E. Testa, M. Soeken, O. Zografos and L. Amarù. *NANOARCH,* Bejing, July 18-20, 2016, pp.1-6.
[3] L. Amarù, P. Gaillardon and G. D. Micheli. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, 2016, pp. 806-819.
[4] A. Mishchenko, S. Chatterjee and R. Brayton. *DAC*, San Francisco, July 24-28, 2006, pp. 1-6.
[5] W. Haaswijk, M. Soeken, L. Amarù, P. Gaillardon and G. D. Micheli. *ASPDAC*, Chiba, January 16-19, 2017, pp. 151-156.
[6] Z. Li, Z. Chu, Y. Xia and L. Wang. *ICSICT*, Qingdao, October 31-November 3, 2018, pp.1-3.