EA-based refactoring of mapped logic circuits

Jitka Kocnova and Zdenek Vasicek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno, Czech Republic
Email: ikocnova@fit.vutbr.cz, vasicek@fit.vutbr.cz

***Abstract*—The increasing complexity of the designs and problematic scalability of original representations led to a shift in internal representations used in logic synthesis and optimization. Heterogeneous representations were replaced with homogeneous intermediate representations. And-inverter graph (AIG) has been identified as the most promising structure for scalable logic optimization and many efficient algorithms were implemented on top of it. However, the inability of AIG to efficiently represent XOR gates together with heuristic nature of logic optimization algorithms leads to some inefficiency causing that the logic can be further minimized even after it has been mapped. This paper presents an optimization technique based on refactoring targeting mapped combinational circuits. It iteratively selects large cones of logic, optimizes them and returns them back to the original structure provided that there is an improvement in some metric. Performance of the method is evaluated on a set of complex academic and industrial benchmarks. We show that a 9.2% reduction in area can be achieved in average compared to the highly optimized results obtained using the academic state-of-the-art synthesis tool. In average, more than 14% reduction was observed for arithmetic circuits.**

I. Introduction

The goal of the logic optimization is to transform a suboptimal solution into an optimal gate-level implementation w.r.t. given synthesis goals, while technology mapping transposes it onto its best standard cell implementation. The circuit is typically represented by a suitable internal representation during the logic optimization. Current state-of-the-art logic synthesis tools, such as ABC, represent circuits using a directed acyclic graph composed of two-input AND nodes denoted as and-inverter graph (AIG). This representation is simple and scalable, and leads to simple algorithms. The optimization of AIGs is based on *rewriting* algorithm which minimizes size of AIG by iteratively selecting subgraphs rooted at a node and replacing them with smaller precomputed subgraphs [1].

Unfortunately, the AIGs suffer from an inherent bias in representation. While eight of ten possible two-input logic gates may be represented by means of a single AIG node, XOR/XNOR gates require three AIG nodes each. The efficiency of synthesis is then limited as it mostly fully relies on transformations disallowing to increase the number of AIG nodes. Also, the synthesis algorithms typically do not treat XORs explicitly – they rely on identification of XORs during the technology mapping phase which works independently on the logic optimization phase. The ability to capture XOR gates is, however, essential for efficient representation of arithmetic and XOR-intensive circuits [2].

To address this problem, e.g. binary decision diagrams (BDDs) can be employed [3], [4]. Due to their limited scalability, Amaru et al. employed a two step synthesis process based on a selective and distinct manipulation of AND/OR and XOR-intensive portions of the logic circuit [5]. Fiser et al. introduced XOR-AIGs to explicitly support XOR gates [6]. Haaswijk et al. employed XOR majority graphs (XMGs) to extend the capabilities of exact synthesis oriented on area optimization.

Other authors tried to avoid intermediate representation. Optimization based on a variant of Genetic Programming (GP) conducted directly at the level of common gates is able to provide significantly better results compared to the state-of-the-art synthesis operating on AIGs [7]. Optimization is done implicitly without any structural biases. In average, the method enabled a 34% reduction in gate count on an extensive set of IWLS benchmark circuits when executed for 15 minutes. A similar approach was successfully applied even to synthesis of conventionally hard to synthesize circuits [8]. The proposed method is able to optimize the circuits for which conventional synthesis completely fails. However, the efficiency of the evolutionary approach deteriorates with the increasing number of gates because of various scalability issues inevitably connected with the usage of GP.

The recent methods need to perform a preprocessing or circuit decomposition [5] or precomputation of ideal solutions[6]; other methods rely on XOR-avare transformations or presence technology cells (eg. XMGs). In order to eliminate the need for circuit preprocessing, we propose a novel logic synthesis methodology that implicitly targets XOR-intensive logic circuits.

II. Background

All circuits can be represented by a *Boolean network* – a directed acyclic graph (DAG) with nodes represented by Boolean functions [9]. The sources are the primary inputs (PIs) of the network and the sinks are the primary outputs (POs). The output of a node may be an input to other nodes called *fanouts*. The inputs of a node are called *fanins*. An edge connects two nodes in fanin/fanout relationship.

A. Limiting the scope of Boolean networks

Network scoping is a key operation to ensure a good scalability of synthesis tools when working with large Boolean networks. *Windowing* and *cut computation* have been proposed to limit the scope of logic synthesis to work only on a small portion of a Boolean network [9].

The windowing algorithm takes a node and two integers: the number of logic levels on the fanin/fanout sides of the node to be included in the window. Leaf set and root set are produced. The window is the subset of nodes of the network containing nodes from root set together with all nodes on paths between the leaf set and the root set. The nodes in the leaf set are not included in the window [9]. It is hard to predict how many logic levels have to be traversed to get a window of the desired size and required number of leaves. Hence, an approach based on k-feasible cuts is preferred. A cut of a node (root node) is a set of nodes of the network (leaves), such that each path from PI to the root node passes through at least one leaf. A cut is k-feasible if the number of nodes (i.e. cut size) in the cut does not exceed k. The volume of a cut is the total number of nodes encountered on all paths between the root node and the cut leaves. To maximize the cut volume, a reconvergence-driven heuristic is applied. The problem is that the cut computed using a naive breadth-first-search algorithm may include only few nodes and leads to tree-like logic structures that do not lead to any don't cares in the local scope of the node and attempting optimization using such a cut would be wasted time [9].

A simple and efficient cut computation algorithm producing a cut close to a given size while heuristically maximizing the cut volume and the number of reconvergent paths subsumed in the cut has been introduced in [9]. Our work is based on the reconvergence-driven cuts and we discuss this algorithm more in the section III.

B. Synthesis of Boolean networks using EAs

Evolutionary algorithms (EAs) have been used to synthesize logic circuits since late nineties [10], [11]. Miller et al., the author of Cartesian Genetic Programming (CGP) [11], is considered as a pioneer in the field of logic synthesis of gate-level circuits. Despite of many advantages of this technique, only small problem instances were typically addressed. The scalability of CGP has been significantly improved by a SAT-based CGP simulator driven by counterexamples produced by the SAT solver [12] [7]. In this area, a linear form of CGP is preferred today. CGP models a candidate circuit having n_i PIs and n_o POs as a linear 1D array of n_n configurable nodes. Each node has n_a inputs and corresponds with a single gate with up to n_a inputs. To avoid a feedback, the inputs can be connected either to the output of a node placed in the previous L columns or directly to PIs. The function of a node can be chosen from a set of n_f functions. Depending on the function of a node, some of its inputs may become redundant. Moreover, the fixed number of nodes n_n does not mean that all the nodes contribute to the POs. These key features allow redundancy and flexibility of CGP. For details of candidate circuits encoding, please see [11].

CGP is a population oriented approach operating with $1+\lambda$ candidate solutions. The initial population is seeded by the original circuit. Every new population contains the best circuit from the previous population, that has not served as a parent yet and its λ offsprings created using a mutation operator that randomly modifies up to h integers. Selection of the individuals is typically based on a cost function (e.g. number of active nodes). Considering the CGP encoding, a single mutation causes either reconnection of a gate, reconnection of primary outputs or change in function of a gate. This procedure is typically repeated for a predefined number of iterations.

III. The proposed method

Let $\mathcal{C}$ be a combinational circuit described at the level of common gates represented by a Boolean network N consisting of $|N|$ nodes. Each node corresponds with a single gate in $\mathcal{C}$. The pseudo-code of the proposed optimization procedure based on evolutionary resynthesis is shown in Algorithm 1.

Algorithm 1: EA-BASED REFACTORING

```
Input: A Boolean network $N$, maximum cut size $cutsize$
Output: Optimized network $N'$, $cost(N') \leq cost(N)$
1  $N' \leftarrow N$
2  while terminated condition not satisfied do
3      $m \leftarrow$ identify the best candidate root node $m \in N'$
4      $C \leftarrow$ ReconvergenceDrivenCut($m$, $cutsize$)
5      $W \leftarrow$ ExpandCutToWindow($m$, $C$)
6      if $W$ is not a suitable candidate then
7          continue
8      $W' \leftarrow$ OptimizeNetworkUsingEA($W$)
9      if cost($(N' \setminus W) \cup W'$) < cost($N'$) then
10         $N' \leftarrow (N' \setminus W) \cup W'$
11 return $N'$
```

Firstly a node which may lead to the best improvement of N is determined. Identification of this node is a nontrivial problem, so some heuristic needs to be implemented – the size of transitive fan-in cone, level of the node or a more complex information can be used. A window is then extracted from the Boolean network. This procedure starts with computation of the reconvergence-driven cut (see Section II-A) and is followed by expansion of the cut C into a window W. In addition to the nodes inside the cut, we consider also all nodes that are not contained in the cut but have fanins inside the cut. Our expansion is similar to that employed in the resubstitution [9] where transitive fanout of C is considered, but we do limit the number of included nodes or their maximum level.

Resynthesis is then applied to the window. Each window potentially leading to no improvement is skipped in order to eliminate execution of a relatively time-consuming resynthesis. Identification of suitable windows can be based on the size of W or a combination of size of C and W (small and thin windows are skipped). We can also use the information about the difference among level of the root node and leaves of C.

The expansion leads to the set of internal nodes I, the set of leaves L and the set of root nodes R. L contains nodes serving as PIs of the temporary network used in the subsequent optimization. R contains nodes whose outputs have to be connected to POs. R contains the root node m and also other nodes with fanouts outside of the window. It holds that $C \subseteq L$ since the expansion may cause that some leaves of C become a fanout of a node inside the window. Two situations can happen for a leaf node. If all fanins are inside the window, the leaf can be simply removed from L. Otherwise, all fanins of the original leaf node need to be added to L. This procedure is repeated iteratively to ensure that there are no leaves having a fanin already included the window.

The resynthesis is performed by means of the CGP. The evolutionary optimization is executed for a limited number of iterations.The more iterations are allowed, the higher improvement can be achieved. However, many iterations on a small window mean a waste of time. Finally, the optimized logic network W' is evaluated w.r.t. N' and if it performs better, it replaces all non-leaf nodes included in W. The whole optimization algorithm is terminated when a predefined number of iterations or a given runtime is exhausted.

IV. Experimental evaluation

A. Experimental setup

Our goal is to evaluate performance of the proposed method w.r.t. the state-of-the-art EA-based method (denoted as global) applied to the whole Boolean network and to compare both methods to the best result produced by the ABC. Both methods operate at the level of optimized and mapped Boolean networks to avoid the bias of AIG representation. The procedure OptimizeNetworkUsingEA is based on the CGP implemented as described in Section II-B with following parameters: $n_a = 2$, $\lambda = 1$, $h = 2$, $n_n = |W|$. A single call of this procedure is executed for the global method. On contrary, several calls of this procedure are executed in the proposed method. The global method terminates after n_{iters} iterations. The proposed method uses a simple divide-and-conquer strategy. The proposed method is allowed to create n_{cuts} cuts. For each cut, the OptimizeNetworkUsingEA is allowed to perform n_{iters}/n_{cuts} iterations. In total, n_{iters} evolutionary iterations are evaluated in both cases. This naive strategy supposes that the computation effort does not depend on the window size but it helps to fairly evaluate the impact of the proposed method. In this paper, we use $n_{iters} = 10^9$ iterations. The $cutsize$ limit is set to 10^4. Only windows with more than 10 nodes are accepted. The root node m is chosen randomly.

This setup was considered the best amongst other setup combinations of the $cutsize$ (5, 10, 20, 35, 50, 75, 100, 150, 200, 250, 10^3, 10^4) and n_{iters} ($10^3, 10^5, 10^7, 10^9$). Experimental results showed a convergence of the number of removed cells after approximately $n_{iters} = 10^7$. $cutsize = 10^4$ cells ensures the possibility of the biggest possible cut creation w.r.t the root cell placement.

This strategy simplifies the problem but may lead to degradation of the performance if many unacceptable windows are produced. The only criterion we consider is the area on a chip expressed as the number of gates. For each method and each benchmark, five independent runs were executed to obtain statistically significant results.

B. Results

The proposed method was implemented in C++ and integrated in Yosys open synthesis suite. Tab. I summarizes the experimental results. The goal was to improve the size of mapped benchmark circuits optimized at the level of AIG by ABC. In particular, we took 19 highly optimized circuits from IWLS'05 Open Cores benchmarks and 9 highly optimized large arithmetic circuits[1].

The circuits were mapped to gates using a library of common 2-input gates including XORs/XNORs (ABC: 'map'). After mapping, optimization by the proposed and global method was executed and final number of mapped gates in circuits was examined. The circuits were then transformed to AIG representation (ABC: 'strash') and compared to the results from ABC-only optimization. All of the optimized circuits were formally verified w.r.t their original form (ABC: 'cec').

Many iterations of resyn script were applied in ABC on the original verilog benchmarks as described in [13] in order to obtain the best results for AIG optimization.

The first three columns of Tab. I contain information related to the benchmarks (name, number of PIs and POs). The next two columns contain number of nodes and depth of circuits in the AIG form after application of ABC resyn script. The other two columns show parameters of the mapped circuits and those numbers serve as a baseline for our comparison – the number of gates and logic depth is provided. Then, the achieved results expressed as the relative reduction w.r.t. the baseline are reported for the proposed and global method. For each method, we report the average and the best obtained improvement. These numbers are calculated from five independent runs.

The best results are very close to the average ones which suggests that the both EA-based methods are stable although they are in principle non-deterministic. According to the number of highlighted cases showing the better results, the proposed method performs substantially better considering the average as well as the best results. It wins in 22 out of 28 cases. The average reduction on the IWLS'05 benchmarks is slightly better in favor of the global method, but it is affected mostly by five cases where the global method provides substantially better results. Looking at the arithmetic circuits, the global method is able to slightly improve only two circuits. In other cases, the reduction is negligible. We analyzed the five cases where the global method outperformed the proposed one and concluded that the global method works well especially for small instances (less than 10^4 gates) that have a reasonable depth (10 to 25 levels). The global optimization of circuits with large depth performs unsatisfactory. Compared to ABC, a substantial improvement is achieved on the arithmetic circuits. The number of gates is reduced by nearly 15% in average. The highest reduction, 30.1%, is recorded for hamming benchmark. The detailed analysis revealed that this was possible due to better handling of XORs/XNORs and also by a relatively huge redundancy of the original circuit optimized by ABC. The relative number of AND/OR/NAND/NOR gates remained nearly the same (around 74%). The number of XORs/XNORs increased from 10% to 15%.

Our second experiment evaluates efficiency of the AIG representation. The last two columns of Tab. I show what happens when we convert the optimized gate-level netlists to AIGs. This section contains the relative size improvement for

[1]All the benchmarks are taken from https://lsi.epfl.ch/MIG

TABLE I: Comparison of the proposed and global method (sec. Impr. proposed, Impr. global) w.r.t. the initial number of mapped gates (sec. ABC(mapped)) and the best result of ABC (sec. ABC(AIG)). Section ABC(AIG) / ABC(mapped) contains parameters of the optimized circuits before and after mapping (D is logic depth, G is the number of gates). Last section shows the size of AIG(relative to ABC) when the gate-level circuit is mapped back to AIG.

			ABC(AIG)		ABC(Mapped)		Impr. proposed		Impr. global [7]		Impr. at AIG level	
Benchmark	PIs	POs	AIG	D	gates	D	gates avg	gates best	gates avg	gates best	proposed	global [7]
DSP	4223	3792	39958	41	43491	45	**3.6%**	**3.6%**	0.0%	0.0%	**0.1%**	0.0%
ac97_ctrl	2255	2136	10497	9	11433	10	**2.9%**	**2.9%**	1.4%	1.4%	0.7%	**1.3%**
aes_core	789	532	20632	19	21128	20	**2.9%**	**2.9%**	0.6%	1.7%	-0.8%	-0.3%
des_area	368	70	5043	24	5199	25	**6.0%**	**6.1%**	2.1%	2.3%	**3.6%**	1.0%
des_perf	9042	1654	75561	15	78972	16	**1.8%**	**1.8%**	0.0%	0.1%	-2.7%	-6.8%
ethernet	10672	10452	56882	22	60413	23	**0.5%**	**0.5%**	0.0%	0.0%	**0.1%**	-0.1%
i2c	147	127	1009	10	1161	12	9.2%	9.2%	**10.0%**	**10.7%**	4.8%	**8.1%**
mem_ctrl	1198	959	9351	22	10459	24	7.0%	7.0%	**24.8%**	**25.4%**	2.4%	**26.0%**
pci_bridge32	3519	3136	16812	18	19020	21	**3.5%**	**3.5%**	0.5%	0.6%	0.4%	**0.5%**
pci_spoci_ctrl	85	60	994	13	1136	15	18.3%	18.5%	**34.8%**	**35.7%**	13.4%	**33.0%**
sasc	133	123	657	7	746	8	**6.2%**	**6.2%**	2.4%	2.8%	0.0%	-0.2%
simple_spi	148	132	770	10	822	11	**5.5%**	**5.7%**	4.4%	4.6%	**1.1%**	0.8%
spi	274	237	3430	24	3825	26	5.6%	5.6%	**13.5%**	**20.2%**	1.7%	**16.0%**
ss_pcm	106	90	381	6	437	7	**5.7%**	**6.7%**	2.3%	2.3%	-0.3%	**0.3%**
systemcaes	930	671	11014	31	11352	27	**11.9%**	**12.3%**	0.0%	0.0%	**3.3%**	-0.4%
systemcdes	314	126	2495	21	2601	25	4.8%	5.0%	**9.1%**	**9.9%**	2.2%	**5.2%**
tv80	373	360	7838	35	8738	39	6.6%	6.9%	**11.1%**	**11.3%**	2.9%	**12.4%**
usb_funct	1860	1692	13914	20	15405	23	**5.8%**	**5.9%**	2.6%	2.6%	1.4%	**2.8%**
usb_phy	113	73	380	7	452	9	**13.9%**	**14.0%**	12.2%	12.2%	3.9%	**5.8%**
average (IWLS'05 benchmarks)			14611	18	15620	20	6.4%	6.5%	**7.0%**	**7.6%**	2.0%	**5.5%**
mult32	64	64	8903	40	8225	42	**16.5%**	**16.6%**	0.0%	0.0%	-1.5%	0.0%
sqrt32	32	16	1353	292	1462	307	**22.3%**	**24.3%**	3.0%	3.0%	**4.2%**	-4.2%
diffeq1	354	193	21980	235	20719	218	**11.5%**	**11.5%**	0.0%	0.0%	**0.7%**	-7.3%
div16	32	32	5111	132	5847	152	**15.7%**	**15.8%**	0.0%	0.0%	**2.1%**	-12.0%
hamming	200	7	2607	73	2724	80	**28.6%**	**30.1%**	14.6%	14.6%	**11.0%**	-0.6%
MAC32	96	65	9099	54	7793	55	**7.7%**	**7.8%**	0.0%	0.0%	-9.7%	-13.0%
revx	20	25	7516	162	8131	171	**14.5%**	**14.5%**	0.0%	0.1%	**1.2%**	-13.0%
mult64	128	128	26024	186	21992	190	**7.4%**	**7.4%**	0.3%	0.5%	-5.4%	-1.0%
max	512	130	2964	113	3719	117	**5.3%**	**5.3%**	0.7%	0.8%	**0.8%**	-0.4%
average (arithmetic benchmarks)			9506	143	8956	148	**14.4%**	**14.8%**	2.1%	2.1%	**0.4%**	-5.7%

the best results produced by the proposed and global method w.r.t. size of the AIGs produced by ABC. We can see that the average reduction is substantially lower compared to the reduction achieved on the gate level representation. In many cases, the AIG of the optimized circuit is even larger than the original one. However, such a behavior is expectable because this happens if the number of XORs increases but the overall number of removed gates is relative small. On the other hand, when the reduction at the level of gates exceeds a certain level the reduction is visible also on AIGs. This is evident especially on the IWLS benchmarks where the global method produces solutions that clearly dominate. From the perspective of AIGs, the global method completely failed on arithmetic benchmarks. The number of AIG nodes substantially increased in almost all cases. As discussed in the introduction, this simple comparison demonstrates the limited capabilities of otherwise efficient AIG representation.

Tab. II shows the average number of leaves, roots and volume of windows produced by the windowing algorithm on some benchmarks. Despite using a simple selection strategy, the parameters are relatively good. The number of leaves $|L|$ determining the number of primary inputs of the refactored subcircuit is substantially higher compared to the sizes of cuts used during rewriting. Number of cut nodes is also satisfactory. Compared to rewriting, a relatively complex portions of the original circuits are chosen for subsequent optimization. This could explain the reason, why the proposed method is able to achieve such reduction. Detailed analysis revealed that the size of the windows is typically higher for the arithmetic circuits.

TABLE II: Average parameters of all windows and windows that led to a reduction (col. successful windows) generated during the refactoring.

	all windows			successful windows		
Benchmark	$\|L\|$	$\|R\|$	size	$\|L\|$	$\|R\|$	size
mem_ctrl	27	25	38	28	26	44
pci_spoci_ctrl	14	13	21	18	19	32
systemcaes	22	15	35	14	13	26
mult32	20	16	34	26	21	52
sqrt32	33	29	62	20	17	37
diffeq1	30	27	53	28	26	55
div16	32	28	50	25	24	44
hamming	30	26	44	26	24	45

V. Conclusion

Compared to the conventional logic synthesis, state-of-the-art EA-based optimization is able to produce substantially better results at the cost of a higher run time that grows with the increasing complexity of the Boolean networks. This paper addresses this problem by combining the EA-based optimization with refactoring that allows to work on a smaller portions of the original Boolean network. Despite using a very simple strategy of root node selection which may degrade the capabilities of the refactoring, the proposed method is able to outperform the AIG-based as well as the original EA-based optimization applied to the whole Boolean networks.

Acknowledgments

This work was supported by Czech Science Foundation project 19-10137S.

References

[1] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting: a fresh look at combinational logic synthesis," in *2006 43rd ACM/IEEE Design Automation Conference*, July 2006, pp. 532–535.

[2] P. Fiser and J. Schmidt, "The observed role of structure in logic synthesis examples," in *18th Int. Workshop on Logic and Synthesis*, 2009, pp. 210–213.

[3] C. Yang and M. Ciesielski, "BDS: a BDD-based logic optimization system," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 7, pp. 866–876, Jul 2002.

[4] N. Vemuri, P. Kalla, and R. Tessier, "Bdd-based logic synthesis for lut-based fpgas," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 4, pp. 501–525, Oct. 2002.

[5] L. Amaru, P. E. Gaillardon, and G. D. Micheli, "Mixsyn: An efficient logic synthesis methodology for mixed xor-and/or dominated circuits," in *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013, pp. 133–138.

[6] P. Fiser, I. Halecek, and J. Schmidt, "Sat-based generation of optimum function implementations with xor gates," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 163–170.

[7] Z. Vasicek, "Cartesian GP in optimization of combinational circuits with hundreds of inputs and thousands of gates," in *Proceedings of the 18th European Conference on Genetic Programming – EuroGP*, ser. LCNS 9025. Springer International Publishing, 2015, pp. 139–150.

[8] P. Fiser, J. Schmidt, Z. Vasicek, and L. Sekanina, "On logic synthesis of conventionally hard to synthesize circuits using genetic programming," in *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010, pp. 346–351.

[9] A. Mishchenko and R. Brayton, "Scalable logic synthesis using a simple circuit structure," in *Int. Workshop on Logic and Synthesis*, 2006, pp. 15–22.

[10] J. D. Lohn and G. S. Hornby, "Evolvable hardware: Using evolutionary computation to design and optimize hardware systems," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 19–27, 2006.

[11] J. Miller and P. Thomson, "Cartesian Genetic Programming," in *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*, ser. LNCS, vol. 1802. Springer, 2000, pp. 121–132.

[12] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 305–327, 2011.

[13] L. Amaru, P. E. Gaillardon, and G. D. Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2016.