THLR: A Top-down Hierarchical Logic Rewrite Framework for Xor-Majority-Inverter Graphs

Ran Zhang[1,2], Rongliang Fu[3*], Shuo Ren[3], Wenxing Li[1], Da Wang[1*], Fei Wang[4], Xiaochun Ye[1], Tsung-Yi Ho[3], Junying Huang[1]

[1]*SKLP, Institute of Computing Technology, CAS*, Beijing, China
[2]*School of Computer Science and Technology, University of Chinese Academy of Sciences*, Beijing, China
[3]*Department of Computer Science and Engineering, The Chinese University of Hong Kong*, Hong Kong, China
[4]*School of Integrated Circuits, Shandong University*, Jinan, China

ABSTRACT

With the increasing complexity of integrated circuits, multiple Boolean network types have been developed to support efficient logic rewriting methods. Although the Xor-Majority-Inverter Graphs (XMG) have a relatively compact expressive power, due to the characteristics of the rewriting itself and the inherent properties of XMG, the rewriting does not always perform optimally in terms of optimization performance on XMG. In this paper, we propose a novel top-down hierarchical logic rewriting framework for XMG that exploits the complementary expressive capabilities of multiple Boolean network types. To more effectively leverage the rewriting potential of hierarchical Boolean network types, we propose a type-aware partitioning strategy that decomposes the network into structurally meaningful sub-circuits. This enables targeted optimizations tailored to the structural characteristics of each sub-circuit, effectively balancing rewriting quality with computational efficiency. Experimental results demonstrate that our framework significantly improves circuit quality, achieving an approximate 4.31% reduction in node-depth product (NDP) compared to state-of-the-art rewriting methods, while also reducing the runtime by about 13.62%. Moreover, after ASIC mapping, THLR delivers a 3.10% improvement in area-delay product (ADP) over the state-of-the-art approaches.

KEYWORDS

Logic Rewriting, Xor-Majority-Inverter Graphs, Graph Partition

ACM Reference Format:
Ran Zhang[1,2], Rongliang Fu[3*], Shuo Ren[3], Wenxing Li[1], Da Wang[1*], Fei Wang[4], Xiaochun Ye[1], Tsung-Yi Ho[3], Junying Huang[1]. 2026. THLR: A Top-down Hierarchical Logic Rewrite Framework for Xor-Majority-Inverter Graphs. In *Great Lakes Symposium on VLSI 2026 (GLSVLSI '26), June 22–24, 2026, Canandaigua, NY, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3787109.3815207

1 INTRODUCTION

With the continuous increase in chip scale, efficient electronic design automation (EDA) tools have become indispensable for successful chip design, as they significantly reduce design cycles and accelerate time-to-market [17, 18]. Logic synthesis [8, 11, 23], a pivotal stage in the EDA frontend flow, plays a fundamental role in chip design. This process comprises two primary stages: logic optimization and technology mapping. Logic optimization abstracts circuit functionality into Boolean networks, independent of physical implementation, while technology mapping translates these abstractions into circuit realizations based on a given cell library.

During the logic optimization phase, Boolean networks can be represented in various forms. Prominent representations include and-inverter graphs (AIG) [13], majority-inverter graphs (MIG) [4], xor-and-inverter graphs (XAG) [14], and xor-majority-inverter graphs (XMG) [12]. Among these, XMG have attracted increasing attention for their strong expressiveness and flexibility, which make them particularly suitable for emerging paradigms such as quantum computing [21] and in-memory computing (IMC) [10, 16].

Considering the inherent computational complexity of logic optimization, deriving globally optimal solutions for arbitrary Boolean network types is generally intractable (NP-hard). Consequently, heuristic logic optimization techniques are widely adopted due to their practical effectiveness and robust empirical performance. Among these, rewriting [5, 7, 9, 20] stands out as a classic logic optimization method that simplifies Boolean networks by extracting sub-circuits and applying localized transformations. Interestingly, although some Boolean network types possess superior theoretical expressive power, rewriting may yield less compact networks compared to those generated from types with lower expressiveness. The implementation mechanism of rewriting, along with its inherently greedy nature, limits its optimization capability. In practice, rewriting expands sub-circuits and searches for more compact replacements. Although the XMG is more expressive, it does not necessarily expose more replaceable sub-circuits during optimization, which constrains the effectiveness of rewriting in this representation. MixSyn [3] exemplifies efforts to exploit multiple structural representations during optimization. Its XOR-prioritizing strategy relies on assumptions about logic type preferences that do not generalize well across diverse circuit types, thus limiting its applicability.

Table 1. Atomic logic operations of various Boolean network types.

Type	Operations
AIG	AND, NOT
MIG	MAJ, NOT
XAG	XOR, AND, NOT
XMG	XOR, MAJ, NOT

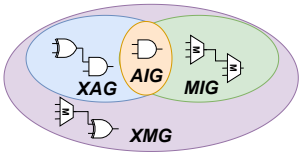

Figure 1. Relationships among various Boolean network types.

* Corresponding authors: rlfu@cse.cuhk.edu.hk, wangda@ict.ac.cn.

GLSVLSI '26, June 22–24, 2026, Canandaigua, NY, USA

ACM ISBN 979-8-4007-2431-2/2026/06
https://doi.org/10.1145/3787109.3815207

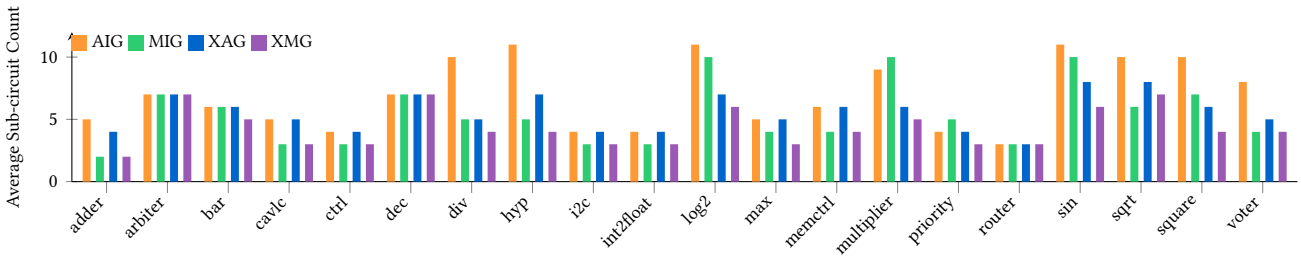

Figure 2. Average number of extended sub-circuits of each vertex in EPFL [2] benchmarks.

In this paper, we propose a novel top-down hierarchical logic rewrite framework for XMG that comprehensively exploits the optimization potentials of diverse Boolean network representations. Specifically, this paper makes the following contributions.

- We propose a top-down hierarchical rewriting framework for XMG that leverages the complementary advantages of various network representations.
- We introduce a type-aware fast partitioning strategy that effectively decomposes the Boolean network into sub-circuits, enabling targeted rewriting techniques tailored to each sub-circuit's structural characteristics.
- We evaluate THLR on selected circuits from the EPFL [2] and IWLS'05 [1] benchmarks. In terms of logic optimization results, THLR achieves a reduction of approximately 4.31% in NDP compared to the best state-of-the-art (SOTA) rewriting approaches. Moreover, after ASIC mapping with ASAP7 [22], THLR delivers a 3.10% improvement in area-delay product (ADP) over the SOTA.

2 PRELIMINARIES

2.1 Boolean Network

Boolean networks are typically classified into four types: AIG, MIG, XAG, and XMG, based on the distinct logical functions represented by their vertices. As shown in TABLE 1, each Boolean network type is defined by a set of atomic logic operations: NOT ($\neg a$), AND ($a \wedge b$), XOR ($a \oplus b$), and MAJ ($\mathrm{M}(a,b,c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$). Specifically, AIG is composed of AND and NOT operations; MIG utilizes MAJ and NOT operations; XAG employs XOR, AND, and NOT operations; and XMG incorporates XOR, MAJ, and NOT operations. Notably, MIG and XAG exhibit greater expressiveness than AIG, since MAJ logic can implement AND by fixing one input to 0, i.e., $\mathrm{M}(a,b,0) = a \wedge b$. Based on this property, a containment relationship among these Boolean networks can be established as illustrated in Fig. 1. From the perspective of expressive power, XMG is the most expressive, followed by XAG and MIG, both of which are strictly more expressive than AIG, yielding the inclusion hierarchy AIG $\subset$ MIG / XAG $\subset$ XMG.

2.2 Logic Rewriting

Logic rewriting [5, 7, 20] is one of the logic optimization techniques that optimize circuits by partially replacing logic cones with functionally equivalent but more compact structures. [7] proposed a rewriting approach that extensively exploits don't-care conditions to achieve substantial circuit size reductions.

To better illustrate our motivation, we applied the rewriting approach described in [7] to each circuit in the EPFL benchmark suite [2] and recorded the number of sub-circuits extended from each vertex. The average counts are summarized in Fig. 2. The results show that, among all representations, XMG-based rewriting approach generate the fewest sub-circuits on average, thereby restricting the number of available replacement opportunities. While this characteristic can accelerate the rewriting process, it also limits the potential for further optimization. These observations indicate that incorporating additional network representations into XMG-based rewriting is both feasible and potentially beneficial.

3 PROBLEM FORMULATION

3.1 Terminology

A Boolean network can be modeled as a directed acyclic graph (DAG), typically denoted as $G(V,E)$, where V represents the set of vertices, and E is the set of directed edges. The vertex set V comprises primary inputs (PIs), primary outputs (POs), and internal nodes, while the directed edge set $E \subseteq V \times V$ illustrates their connectivity. The orientation of these edges signifies the direction of data flow within the circuit. For any vertex $v \in V$, the fan-in set $\mathrm{FI}(v)$ and fan-out set $\mathrm{FO}(v)$ are defined as the sets of vertices with edges terminating at v and originating from v, respectively. Specifically, for any $u \in \mathrm{FI}(v)$, there exists an edge $(u,v) \in E$; similarly, for any $v \in \mathrm{FO}(u)$, there exists an edge $(u,v) \in E$. It is noteworthy that PIs (POs) have empty fan-in (fan-out) sets. The neighbors of v, denoted $N(v)$, are collectively defined as $N(v) = \mathrm{FI}(v) \cup \mathrm{FO}(v)$.

3.2 Problem Formulation

The primary objective of the proposed logic rewriting framework is to simultaneously minimize the gate count and logic depth. Formally, the optimization problem is formulated as follows:

- **Input:** A Boolean network $G(V,E)$.
- **Output:** An optimized Boolean network $G'(V',E')$.
- **Goal:** min $\{\mathrm{NDP} = node \times depth\}$, where the node-depth product (NDP) is a composite metric widely used in logic optimization to balance spatial and temporal complexity by combining the number of gates (*node*) and the circuit's logic depth (*depth*), holistically reflecting circuit performance.

3.3 Challenges

The top-down hierarchical logic rewriting framework offers notable potential for improving circuit quality; however, its practical deployment faces several significant challenges that must be addressed to

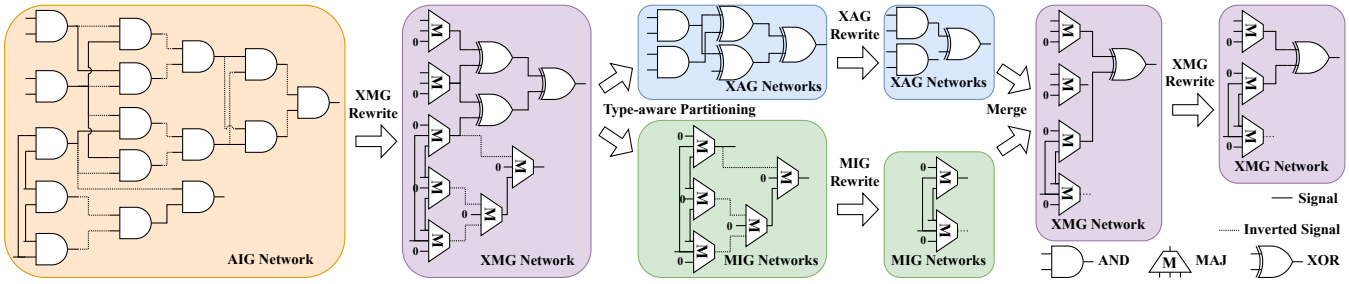

Figure 3. The overall flow of THLR.

facilitate broader adoption. A primary issue concerns which representations are suitable for inclusion in the architecture. While incorporating diverse representations, as discussed in Section 2.2, can enhance the effectiveness of XMG-based rewriting by providing additional alternative structures, introducing too many representations into the framework may also lead to substantial computational overhead, particularly in terms of runtime. Another pressing challenge is the accurate partitioning of Boolean networks. Defining accurate boundaries between partitions is inherently difficult because logic primitives, such as MAJ gates, can often be equivalently represented using alternative formulations, like AND-based logic. Suboptimal partitioning can undermine the effectiveness of later optimization steps and negatively impact overall framework performance.

4 THLR

To address the key challenges outlined in Section 3.3, we introduce THLR, a top-down hierarchical XMG logic rewriting framework, which utilizes the expressive capabilities of hierarchical Boolean networks. Specifically, in Section 4.1, we analyze the properties of various Boolean network representations and investigate which representations are most suitable for integration into THLR to achieve a balanced improvement in rewriting quality while maintaining acceptable computational cost. Subsequently, in Section 4.2, we propose a type-aware partitioning strategy to address the challenge of high-quality circuit partitioning. Together, these sections form an efficient and flexible framework capable of performing effective rewriting across complex Boolean networks.

4.1 Hierarchical Logic Rewriting

As introduced in Section 2.1, the inclusion relationship among Boolean network types follows the hierarchy AIG $\subset$ MIG / XAG $\subset$ XMG. Although employing multiple network representations can potentially improve the optimization quality of XMG rewriting, this strategy inevitably incurs significant computational overhead. To balance rewriting quality with computational efficiency, we designate XMG-based rewriting as the top-level optimization, serving as the foundation for subsequent fine-grained optimizations. Thereafter, MIG-based and XAG-based rewritings are applied at a fine-grained level to sub-circuits obtained from the partitioning method described in Section 4.2. The AIG-based rewriting is excluded due to its limited expressive capability and prohibitive computational overhead.

Fig. 3 shows the overall workflow of THLR. Initially, XMG-based rewriting is applied to transform the circuit into an XMG network using the following identities: $a \oplus b = \neg(\neg(a \wedge \neg b) \wedge \neg(\neg a \wedge b))$ and $\mathrm{M}(a, b, 0) = a \wedge b$. Then, the XMG network is partitioned into XOR and MAJ sub-circuits using the type-aware partitioning strategy detailed in Section 4.2. Sub-circuits identified as XOR are optimized using XAG-based rewriting rules, for example, $b \oplus c = ((a \oplus b) \oplus (a \oplus c))$, while MAJ sub-circuits undergo MIG-based rewriting, such as $\mathrm{M}(a, b, c) = \neg\mathrm{M}(\neg\mathrm{M}(a, b, 0), \mathrm{M}(\neg\mathrm{M}(a, c, 0), \neg\mathrm{M}(b, c, 0), 0), 0)$. This targeted, hierarchical strategy leverages the unique strengths of each Boolean network type to enhance the optimization quality.

To further explore optimization opportunities, the type-aware partitioning strategy in Section 4.2 permits the reassignment of vertices in a Boolean network to atomic logic operation types that are different from their original logical semantics. However, these conversions introduce additional structural overhead. For instance, the transformation of an MAJ vertex into XOR and AND logic incurs additional complexity by introducing some extra vertices, according to the identity $\mathrm{M}(a, b, c) = (a \wedge b) \vee \big(c \wedge (a \oplus b)\big)$. To mitigate this issue, an XMG-based rewriting is applied as post-optimization after the optimized sub-circuits are merged into a unified Boolean network, as illustrated on the right of Fig. 3. This post-processing effectively rectifies suboptimal vertex assignments by restoring them to more appropriate Boolean network types when necessary, thereby yielding a more compact and efficient circuit.

4.2 Type-aware Circuit Partitioning

To address the challenge of determining high-quality partition boundaries, we propose a novel circuit partitioning strategy. After the XMG-based rewriting, the Boolean network comprises only XOR and MAJ vertices, which are widely distributed and intricately intertwined throughout the network. This poses a challenge to distinguishing the boundary between the two networks, because, as mentioned in Section 2.1, some MAJ vertices can be seen as transformations of AND vertices, which can be used in XAG networks for rewriting. As a result, a naive partitioning based solely on vertex type fails to capture meaningful structural information, limiting the effectiveness of subsequent rewriting. To overcome this, our method uses vertex type as an initial partitioning criterion, followed by a refinement step based on the Label Propagation Algorithm (LPA)[19] to reassign vertex types and uncover further optimization opportunities. The reassignment is selectively applied within small partitions to enhance structural coherence while preserving the overall partitioning hierarchy. The complete procedure is summarized in the pseudocode presented in Algorithm 1.

Initially, the Boolean network is divided into two partitions by vertex type. All vertices are flagged as eligible for reassignment (lines 1-4). For those vertices identified as belonging to the XOR partitions, we argue that their intrinsic logical semantics are not amenable

Algorithm 1: Type-aware partitioning algorithm.

Input: An XMG network $G(V, E)$, max iteration count I_{max}.
Output: A partitioning P of vertices.

1 $P \leftarrow \{\text{XOR:}\{\},\text{MAJ:}\{\}\}, C \leftarrow \{true | v \in V\}$
2 **for** $v \in V$ **do**
3 $t(v) \leftarrow$ Get type of v
4 $P(t(v)) \leftarrow P(t(v)) \cup v$
5 **for** $p \in P$ **do**
6 **if** $p \in XOR$ **then**
7 $C(v) \leftarrow false, v \in p$
8 **continue**
9 **for** $v \in p$ **do**
10 $q, s \leftarrow \{v\}$
11 **while** q *not empty* **do**
12 $n \leftarrow q.pop()$
13 $q \leftarrow q \cup \{u | u \in N(n) \wedge u \in p \wedge u \notin s\}$
14 $s \leftarrow s \cup \{u | u \in N(n) \wedge u \in p \wedge u \notin s\}$
15 **if** $|s| \geq \lambda$ **then**
16 $C(v) \leftarrow false$
17 **break**
18 $e \leftarrow true, i \leftarrow 0$
19 **while** e *is true* $\wedge\ i < I_{max}$ **do**
20 $e \leftarrow false$
21 **for** $v \in V$ **do**
22 **if** $C(v)$ *is true* **then**
23 $P(t(v)) \leftarrow P(t(v)) \setminus v$
24 $t(v) \leftarrow \arg\max \sum_{u \in N(v)} \delta(t(u))$
25 $P(t(v)) \leftarrow P(t(v)) \cup v$
26 **for** $v \in V$ **do**
27 **if** $t(v) \neq \arg\max \sum_{u \in N(v)} \delta(t(u))$ **then**
28 $e \leftarrow true$
29 $i \leftarrow i + 1$
30 **return** P

to further transformation or simplification within the considered framework. Consequently, we explicitly define their transformation condition as $C(v) = false, v \in p$ (lines 6–8), thereby excluding them from subsequent transformation procedures. This treatment reflects the inherent compactness of XOR components, for which transformations of logical semantics are generally not considered to preserve their structural efficiency. The algorithm identifies vertices for potential reassignment by maintaining a queue q and a visited vertices set s (lines 9–17). When the size of the visited set reaches a threshold λ, reassignment eligibility flags are updated accordingly (lines 15-17). The type-aware reassignment proceeds iteratively when e is true and iteration count $i < I_{max}$ (lines 19-29). When the eligibility flag of v is true ($C(v) = true$), the vertex type $t(v)$ is updated to the most frequent neighboring label according to:

$$t(v)^{(i+1)} = \arg\max \sum_{u \in N(v)} \delta(t(u)^{(i)}), \tag{1}$$

where $t(v)^{(i+1)}$ denotes the updated label of vertex v at iteration $i+1$. The function $\delta(t(u)^{(i)})$ produces a one-hot vector representation of the label $t(u)^{(i)}$, with a value of 1 at the position corresponding to the label and 0 elsewhere (lines 22-25). For each vertex v, if its current type differs from the majority type among its neighbors $N(v)$, a flag e is raised (lines 26–28). The process terminates when no vertex type changes occur or the maximum number of iterations is reached. An example in Fig. 4 illustrates this: MAJ vertices surrounded mainly by XOR neighbors update their type to AND, enabling XAG-based

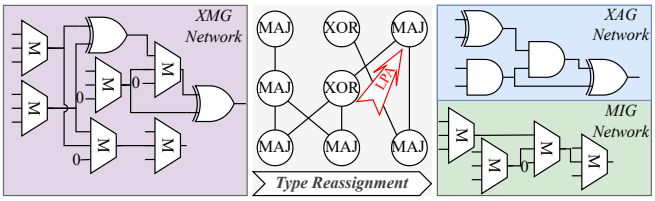

Figure 4. MAJ vertices are reassigned to AND.

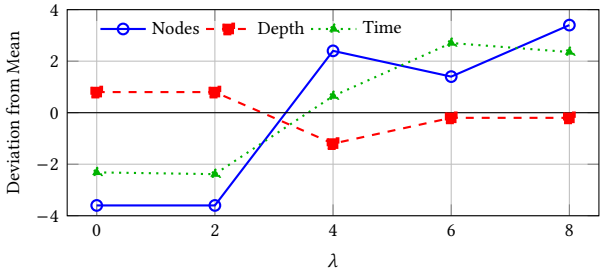

Figure 5. Deviation of node count, depth, and runtime from the mean for the `div` circuit across different λ values.

rewriting of the corresponding sub-circuit. This controlled refinement thus promotes the formation of meaningful sub-circuits and enhances the overall rewriting quality.

In practice, during the iterative addition of vertex u to the set s in Algorithm 1 (lines 11-17), the algorithm terminates once the condition $|s| \geq \lambda$ is satisfied (lines 15-17). This early termination ensures that the vertex eligibility checking phase operates with a time complexity of $O(V)$. For the reassignment procedure, given that the maximum iteration count I_{max} is a constant, its time complexity is similarly bounded by $O(V)$. Consequently, the overall time complexity of the type-aware partitioning strategy is $\max(O(V), O(V)) = O(V)$.

5 EXPERIMENTAL RESULTS

This section presents the optimization results of the proposed THLR framework. The framework was implemented in C++. To avoid uninformative cases, we excluded some EPFL [2] circuits that offer almost the same rewriting results. We also added several large circuits from IWLS'05 [1] to more clearly illustrate the effectiveness of THLR on industrial-scale circuits. All experiments were conducted on a machine equipped with an Intel(R) Xeon(R) Platinum 8350C processor and 1.5 TB of memory. All cases were preprocessed with `resyn2` optimization in the ABC [6] before the experiment.

Existing hybrid optimization methods, such as CHOP [11] and HeLO [15], employ a combination of logic optimization techniques, including rewriting, refactoring, balancing, and resubstitution, rather than relying solely on logic rewriting. This makes them unsuitable as baselines for this study. To evaluate the effectiveness of THLR, we select the state-of-the-art rewrite method [7] for a single Boolean network type, alongside the MixSyn [3], configured with the same logic optimization method as in [7] (XAG as top-level, AIG/XAG as fine-grained, and XMG as post-optimization), for comparative evaluation. For all experiments, we set I_{max} as $\min(depth \times 0.1, 20)$.

The baselines include: (1) single logic representations (AIG, MIG, XMG, XAG), each undergoing two rounds of rewriting followed by an XMG-based rewrite; and (2) MixSyn, which adopts XAG as top-level, AIG/XAG for fine-grained optimization, and XMG for post-optimization.

Table 2. Performance on the EPFL[2] Benchmark with logic rewriting.

Circuit	Origin		AIG^2+XMG[7]			MIG^2+XMG[7]			XMG^3[7]			XAG^2+XMG[7]			MixSyn[3]			THLR		
	node	depth	node	depth	NDP	node	depth	NDP	node	depth	NDP	node	depth	NDP	node	depth	NDP	node	depth	NDP
adder	1019	255	635	128	0.31	391	128	0.19	389	128	0.19	635	128	0.31	635	128	0.31	383	128	0.19
cavlc	662	16	550	14	0.73	551	16	0.83	551	14	0.73	545	15	0.77	548	14	0.72	551	14	0.73
ctrl	108	8	90	7	0.73	92	7	0.75	93	7	0.75	90	7	0.73	90	7	0.73	88	7	0.71
div	40772	4361	16807	2231	0.21	16869	2247	0.21	16857	2229	0.21	16877	2231	0.21	16877	2231	0.21	16790	2228	0.21
DSP	39064	51	30899	52	0.81	31687	50	0.80	30779	51	0.79	31236	48	0.75	31236	48	0.75	31185	48	0.75
ethernet	56152	27	45171	27	0.80	45262	27	0.81	45154	27	0.80	45184	27	0.80	45207	28	0.83	45222	27	0.81
hyp	211330	24794	100755	8910	0.17	100312	9015	0.17	99561	8896	0.17	99718	8907	0.17	99718	8907	0.17	99355	8896	0.17
i2c	1162	15	992	13	0.74	993	13	0.74	998	13	0.74	993	13	0.74	993	13	0.74	979	13	0.73
int2float	214	15	177	13	0.72	177	13	0.72	177	13	0.72	178	13	0.72	178	13	0.72	176	13	0.71
leon2	776253	45	629645	52	0.94	630694	43	0.78	630188	41	0.74	630150	44	0.79	630202	43	0.78	630187	41	0.74
leon3	990072	46	785761	47	0.81	786486	46	0.79	786242	45	0.78	786178	47	0.81	786178	47	0.81	786420	44	0.76
leon3opt	957328	51	779125	51	0.81	780274	42	0.67	780006	41	0.66	779960	42	0.67	779960	42	0.67	779800	41	0.65
leon3mp	582768	42	466978	47	0.90	467550	38	0.73	467304	38	0.73	467134	40	0.76	467320	39	0.74	467002	38	0.73
max	2834	204	1845	148	0.47	1838	148	0.47	1838	148	0.47	1845	148	0.47	1845	148	0.47	1829	148	0.47
memctrl	45614	110	36910	104	0.77	36565	102	0.74	36822	103	0.76	36837	104	0.76	36881	103	0.76	37342	100	0.74
multiplier	24556	265	15936	132	0.32	15912	133	0.33	15639	133	0.32	15946	132	0.32	15946	132	0.32	15897	132	0.32
priority	676	203	425	59	0.18	491	209	0.75	503	208	0.76	425	59	0.18	420	65	0.20	419	58	0.18
router	177	19	145	22	0.95	149	22	0.97	149	22	0.97	147	21	0.92	148	21	0.92	158	17	0.80
sin	5039	177	3641	105	0.43	3647	105	0.43	3628	104	0.42	3615	106	0.43	3617	106	0.43	3612	105	0.43
sqrt	19437	4968	10765	3775	0.42	10767	3715	0.41	8726	2270	0.21	8492	2206	0.19	8492	2206	0.19	8505	2217	0.20
square	16623	248	8215	127	0.25	8164	128	0.25	8082	127	0.25	8281	127	0.26	8281	127	0.26	8059	128	0.25
voter	9756	57	4302	31	0.24	4652	40	0.33	4196	31	0.23	3984	33	0.24	4110	33	0.24	4198	31	0.23
Ave. ratio	1.00	1.00	0.70	0.78	0.58	0.70	0.80	0.59	0.69	0.77	0.56	0.70	0.74	0.55	0.70	0.74	0.55	**0.69**	**0.72**	**0.52**

5.1 Setting of λ

Fig. 5 shows the effect of different λ settings on the logic rewriting results for the `div` circuit. Since λ is a key parameter in type-aware partitioning, increasing its value leads to higher computational time. Conversely, setting λ too small may cause some vertices to be skipped during optimization, while setting it too large can result in excessive conversion of certain vertices to other types, as illustrated in the figure. As shown in the figure, when $\lambda > 4$, both the logic depth of the optimized circuit and the runtime increase significantly, whereas for $\lambda < 4$, the logic depth remains relatively high. Considering that the logic depth of most Boolean networks is much smaller than the number of vertices in typical circuits, we set $\lambda = 4$.

5.2 Result with Logic Rewriting

TABLE 2 presents the rewriting results for using different representations as well as the results of MixSyn and THLR. The results in the table show that applying different representations to rewriting across the entire network yields relatively poor optimization performance. Although some representations (such as XAG) provide better results than using only XMG for rewriting, they still perform slightly worse than the more fine-grained partition-based THLR method.

This highlights the advantage of THLR in exploring a broader optimization space by combining multiple Boolean network representations and their corresponding rewriting approaches in XMG networks. For instance, in the `priority` circuit, THLR produces substantially more compact results than XMG-based rewriting alone by exploiting optimization opportunities unique to other representations. This reflects the flexibility of THLR in adapting to circuit-specific characteristics, an adaptability that is unattainable through the repeated application of the single representation rewriting method. In essence, the strength of THLR lies in the diversity of rewrite pathways it integrates. Overall, THLR achieves average NDP reductions of 9.03%, 10.11%, 5.60%, 4.31%, and 4.48% compared to AIG^2+XMG, MIG^2+XMG, XMG^3, XAG^2+XMG, and MixSyn, respectively, on networks where NDP changes.

Table 3. Mapping results with ASAP7 [22] (Area: μm, Delay: ps).

Circuit	XMG^3[7]		XAG^2+XMG[7]		MixSyn[3]		THLR	
	Area	Delay	Area	Delay	Area	Delay	Area	Delay
div	2388.16	59197.8	2390.61	59072.3	2389.63	59077.1	2389.67	59248.5
DSP	2763.03	1114.93	2801.77	985.05	2801.77	985.05	2809.64	925.14
ethernet	4174.69	523.45	4168.52	437.25	4168.52	437.25	4169.47	414.24
hyp	14571.0	243382	14341.9	242680	13917.6	242513	13673.7	245049
leon2	65991.6	828.41	66169.2	893.04	66028.8	836.63	66044.9	798.42
leon3	76458.1	754.68	76497.4	852.67	76429.2	841.82	76462.3	754.68
leon3opt	76926.2	860.24	76558.8	887.29	76437.8	877.73	76944.7	860.24
leon3mp	43459.8	692.27	43465.5	692.27	43465.5	692.27	43444.8	688.78
memctrl	3302.86	1701.31	3267.7	1695.4	3278.17	1669.52	3327.89	1649.67
Ave. ratio	1.01	1.06	1.00	1.05	1.00	1.04	**1.00**	**1.00**

5.3 Result with ASIC Mapping

Although logic rewriting is technology-independent, its impact is best assessed using technology mapping metrics, which critically influence downstream physical implementation. To further evaluate this impact, we selected the nine largest circuits for ASIC mapping using the ASAP7 [22] and compared the four best-performing rewriting methods identified in Section 5.2 (XMG^3, XAG^2+XMG, MixSyn, and THLR). The corresponding results are presented in TABLE 3. After technology mapping, THLR consistently maintains its advantage, achieving delay-area product (ADP) reductions of 5.36%, 4.53%, and 3.10% over XMG^3, XAG^2+XMG, and MixSyn, respectively.

5.4 Runtime

TABLE 4 summarizes the runtime for using different representations, as well as the runtimes for MixSyn and THLR. Among the results, the method to enhance XMG rewriting performance using AIG-based rewriting incurs the highest computational cost. This justifies the exclusion of AIG from the candidate representation in Section 4.1.

Table 4. Runtime on different rewriting methods.

Time(s)	AIG^2+ XMG[7]	MIG^2+ XMG[7]	XMG^3[7]	XAG^2+ XMG[7]	MixSyn[3]	THLR
adder	9.33	5.98	5.23	5.53	11.27	5.18
cavlc	11.27	4.84	5.05	5.75	7.60	4.94
ctrl	11.20	4.82	4.93	5.57	7.39	4.84
DSP	48.48	27.65	28.17	43.32	32.19	27.33
div	59.71	29.31	22.33	36.29	57.97	24.72
ethernet	61.90	40.37	33.496	57.725	45.86	29.89
hyp	366.11	178.45	178.74	219.66	368.26	226.92
i2c	11.61	5.02	5.30	6.89	7.89	6.84
int2float	9.16	4.78	5.14	5.49	7.43	4.48
leon2	1557.43	1405.06	1343.45	1523.25	1834.25	1601.58
leon3	2073.34	1899.47	1722.27	2051.44	2178.54	1957.43
leon3opt	1804.36	1817.46	1635.84	1820.34	2147.23	2065.52
leon3mp	914.68	777.61	757.93	929.81	1011.55	898.085
max	11.01	6.13	6.81	7.74	10.37	6.18
memctrl	51.67	30.95	28.61	28.68	48.52	33.55
multiplier	54.79	28.89	18.51	30.69	28.22	17.82
priority	11.84	6.49	7.39	6.48	8.20	7.25
router	9.29	5.84	5.81	5.42	7.52	6.64
sin	22.29	10.68	9.79	13.70	14.16	10.67
sqrt	29.55	18.87	18.24	22.74	29.80	16.69
square	43.05	16.30	11.92	24.12	23.21	12.65
voter	21.38	9.88	8.52	13.54	21.413	8.54
Ave. ratio	1.87	1.03	0.95	1.23	1.53	**1.00**

Overall, THLR achieves speedups of approximately 1.87× over AIG^2+XMG, 1.03× over MIG^2+XMG, 1.23× over XAG^2+XMG, and 1.53× over MixSyn, primarily because it adopts XMG as the top-level type, the most efficient type among the four. However, compared to XMG^3 alone, THLR experiences a slight slowdown (0.95×). This demonstrates that THLR achieves a good balance between runtime and optimization quality. Although the improvement in NDP over the XAG^2+XMG method is modest, our approach reduces the runtime by approximately 13.62%. In comparison with XMG^3, our method yields better NDP results, albeit with a longer runtime. The increased runtime relative to XMG^3 is reasonable and can be attributed to two primary factors. First, THLR incorporates additional rewrite steps involving XAG and MIG networks, both of which are inherently more computationally intensive than XMG due to their structural complexity and the nature of their rewrite algorithms. Second, the framework performs circuit partitioning as well as subgraph splitting and recombination, which introduces additional processing stages and results in increased runtime. Notably, the performance gains achieved by THLR in terms of circuit compactness and logic quality outweigh the slight increase in runtime, demonstrating a favorable trade-off between optimization effectiveness and computational cost.

6 CONCLUSION

This paper proposed THLR, a novel top-down hierarchical XMG logic rewriting framework that utilizes multiple representations to enhance optimization quality. To support this, we introduce a type-aware circuit partitioning strategy that divides the Boolean network into structurally meaningful sub-circuits, enabling the framework to exploit the optimization potential of diverse network types while overcoming the limitations of existing rewriting approaches. In terms of NDP, THLR achieves a reduction of approximately 4.31% compared to the best results obtained by SOTA rewriting methods, while simultaneously reducing the runtime by 13.62%. Moreover, after ASIC mapping, THLR further delivers a 3.10% improvement in ADP over the SOTA approaches.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62302477); in part by the State Key Lab of Processors, Institute of Computing Technology, CAS (Grant No. CLQ202402); in part by the Shandong Province Natural Science Foundation, China (Grant No. ZR2024MF073); and in part by the Shandong Province Major Scientific and Technological Innovation Projects, China (Grant No. 2025S0101-01954).

REFERENCES

[1] C. Albrecht. 2005. *IWLS 2005 Benchmarks.* Technical Report. Cadence Research Laboratories at Berkeley. http://www.iwls.org/iwls2005/benchmarks.html
[2] Luca Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL Combinational Benchmark Suite. In *Proc. IWLS.*
[3] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2013. MIXSyn: An efficient logic synthesis methodology for mixed XOR-AND/OR dominated circuits. In *Proc. ASPDAC.* 133–138.
[4] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2014. Majority-Inverter Graph: A novel data-structure and algorithms for efficient logic optimization. In *Proc. DAC.* 1–6.
[5] Alan Mishchenko Robert Brayton and Alan Mishchenko. 2006. Scalable logic synthesis using a simple circuit structure. In *Proc. IWLS*, Vol. 6. 15–22.
[6] Robert Brayton and Alan Mishchenko. 2010. ABC: an academic industrial-strength verification tool. In *Proc. CAV.* Springer-Verlag, 24–40.
[7] Alessandro Tempia Calvino and Giovanni De Micheli. 2024. Scalable Logic Rewriting Using Don't Cares. In *Proc. DATE.* 1–6.
[8] Rongliang Fu, Libo Shen, Ziyi Wang, Zhengxing Lei, Zixiao Wang, Junying Huang, Bei Yu, and Tsung-Yi Ho. 2026. DCLOG: Don't Cares-based Logic Optimization using Pre-training Graph Neural Networks. In *Proc. ASPDAC.* 793–799.
[9] Rongliang Fu, Wei Xuan, Shuo Yin, Guangyu Hu, Chen Chen, Hongce Zhang, Bei Yu, and Tsung-Yi Ho. 2026. eLogic: A E-Graph-based Logic Rewriting Framework for Majority-Inverter Graphs. In *Proc. DATE.* 1–6.
[10] Rongliang Fu, Ran Zhang, Libo Shen, Wei Xuan, Ning Lin, Junying Huang, Bei Yu, and Tsung-Yi Ho. 2026. Efficient Multi-Array Parallel Scheduling for In-Memory Computing. *IEEE TCAD* (2026). https://doi.org/10.1109/TCAD.2026.3661432
[11] Rongliang Fu, Ran Zhang, Ziyang Zheng, Zhengyuan Shi, Yuan Pu, Junying Huang, Bei Yu, Qiang Xu, and Tsung-Yi Ho. 2026. CHOP: Clustered Hybrid Optimization for Logic Synthesis with Self-Supervised Prediction. *IEEE TCAD* (2026).
[12] Winston Haaswijk, Mathias Soeken, Luca Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2017. A novel basis for logic rewriting. In *Proc. ASPDAC.* 151–156.
[13] Leo Hellerman. 1963. A Catalog of Three-Variable Or-Invert and And-Invert Logical Circuits. *IEEE Transactions on Electronic Computers* EC-12, 3 (1963), 198–223.
[14] Giulia Meuli, Mathias Soeken, and Giovanni Micheli. 2022. Xor-And-Inverter Graphs for Quantum Compilation. *npj Quantum Information* 8 (12 2022).
[15] Yuan Pu, Fangzhou Liu, Zhuolun He, Keren Zhu, Rongliang Fu, Ziyi Wang, Tsung-Yi Ho, and Bei Yu. 2025. HeLO: A Heterogeneous Logic Optimization Framework by Hierarchical Clustering and Graph Learning. In *Proc. ISPD.* 116–124.
[16] Xingyue Qian, Chen Nie, Zhezhi He, and Weikang Qian. 2025. MASIM: An Energy-Efficient Multi-Array Scheduler for SIMD Logic-in-Memory Architectures. In *Proc. ICCAD.* 1–9.
[17] Shantian Qin et al. 2025. PANDA: Adaptive Prefetching and Decentralized Scheduling for Dataflow Architectures. 22, 2, Article 62 (July 2025), 27 pages.
[18] Shantian Qin et al. 2025. StreamDCIM: A Tile-based Streaming Digital CIM Accelerator with Mixed-stationary Cross-forwarding Dataflow for Multimodal Transformer. (2025), 1–5.
[19] Aria Rezaei, Saeed Mahlouji Far, and Mahdieh Soleymani. 2015. Near Linear-Time Community Detection in Networks with Hardly Detectable Community Structure. In *Proc. ASONAM.* 65–72.
[20] Heinz Riener, Winston Haaswijk, Alan Mishchenko, Giovanni Micheli, and Mathias Soeken. 2019. On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis. In *Proc. DATE.* 1649–1654.
[21] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. 2017. Design automation and design space exploration for quantum computers. In *Proc. DATE.* 470–475.
[22] Vinay Vashishtha, Manoj Vangala, and Lawrence T. Clark. 2017. ASAP7 predictive design kit development and cell design technology co-optimization: Invited paper. In *Proc. ICCAD.* 992–998.
[23] Ran Zhang, Xinda Chen, Rongliang Fu, Chunyang He, Da Wang, Fei Wang, Tsung-Yi Ho, and Junying Huang. 2026. RECALLS: Reinforcement Learning Enhanced Generative Model for Logic Synthesis Optimization. In *Proc. ISCAS.*