A Don't-Care-Based Approach to Reducing the Multiplicative Complexity in Logic Networks

Hsiao-Lun Liu, Yi-Ting Li, Yung-Chih Chen, and Chun-Yao Wang, *Member, IEEE*

***Abstract*—Reducing the number of AND gates in logic networks benefits the applications in cryptography, security, and quantum computing. This work proposes a don't-care-based (DC-based) approach to reduce the number of AND gates further in the well-optimized network. Furthermore, this work also proposes an enhanced synthesis flow by integrating our approach with the state-of-the-art. The experimental results show that our approach can further reduce up to 25% of the number of AND gates in the network. For the experiments about the enhanced synthesis flow, we achieve a speedup of almost 10× on average for the cryptography benchmarks while having competitive results as compared to the flow in the state-of-the-art.**

***Index Terms*—Don't-care, multiplicative complexity, SAT, XOR-AND graphs (XAGs).**

I. Introduction

TRADITIONALLY, logic optimization aims to optimize timing, area, or power consumption [4], [5], [10]. However, in recent years, an optimization objective regarding minimizing the multiplicative complexity in logic networks for cryptography and security applications has been proposed [18], [19]. In the field of cryptography, logic networks are usually represented with {XOR, AND, NOT}, which form XOR-AND graphs (XAGs) [3]. XAGs can represent any Boolean function because an arbitrary Boolean function f can be represented in Reed–Muller form [16]

$$f(x_1, x_2, \ldots, x_n) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_1x_2 \oplus \cdots \oplus a_{2^n-1}x_1x_2\cdots x_n \quad (1)$$

which is a subset of XAGs. In (1), $a_i \in \{0, 1\}$, $0 \le i \le 2^n - 1$.

The multiplicative complexity of a logic network is defined as the number of AND gates in this network [18]. The motivations of reducing the number of AND gates in a logic network can be classified into two categories. First, in some high-level cryptography protocols, such as zero-knowledge protocols [9], fully homomorphic encryption (FHE), and secure multiparty computation (MPC) [1], AND gates represent the bottleneck of computation as compared to other logic gates [11]. Hence, reducing the number of AND gates in a logic network can decrease the computational overheads. Second, in fault tolerant quantum computing, the execution cost of T gate is significantly large as compared to the other gates [2]. Thus, we only count the number of T gates (T-count) as evaluating the computational cost of a quantum circuit. Meuli *et al.* [14] proposed a constructive method to create quantum circuits with T-count that is at most four times the number of AND gates in the network. Therefore, minimizing the number of AND gates in a logic network bounds the computational cost of a quantum circuit.

Because of the mentioned motivations in various applications, minimizing the number of AND gates in a logic network becomes an optimization objective in logic synthesis. Testa *et al.* [18] proposed a rewriting method taking advantage of affine functions [13] classification for addressing this problem. The state-of-the-art [19] extended [18] by adding refactoring and resubstitution techniques to form a complete synthesis flow.

In this work, we propose a don't-care-based (DC-based) approach, which further minimizes the number of AND gates in well-optimized logic networks. Additionally, we propose an enhanced synthesis flow by integrating our approach with the state-of-the-art. The proposed DC-based approach consists of two techniques. The first one exploits satisfiability don't-cares (SDCs) and observability don't-cares (ODCs) to transform AND gates into XNOR gates. The method of finding SDCs and ODCs is formulated as a SAT problem. The other technique minimizes the number of AND gates in the XAG by merging AND gates or transforming AND gates into other gates considering ODCs [5]–[8]. The experiments were performed on a set of well-optimized EPFL and cryptography benchmarks obtained by previous works [18], [19]. The experimental results show that our approach can further reduce up to 25% of the number of AND gates in the network. The experimental results of the proposed enhanced synthesis flow achieve a speedup of almost 10× on average for the cryptography benchmarks while having competitive results as compared to the flow of reiterating the state-of-the-art method until no gains.

Manuscript received 3 September 2021; revised 23 November 2021; accepted 18 January 2022. Date of publication 28 January 2022; date of current version 24 October 2022. This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 108-2218-E-007-061, Grant MOST 109-2221-E-007-082-MY2, Grant MOST 109-2221-E-155-047-MY2, Grant MOST 109-2224-E-007-005, and Grant MOST 110-2224-E-007-007. This article was recommended by Associate Editor M. Ciesielski. *(Corresponding author: Hsiao-Lun Liu.)*

Hsiao-Lun Liu, Yi-Ting Li, and Chun-Yao Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: hsiaolunliu@gmail.com; yitingli.yt@gmail.com; wcyao@cs.nthu.edu.tw).

Yung-Chih Chen is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 10607, Taiwan (e-mail: ycchen.ee@mail.ntust.edu.tw).

Digital Object Identifier 10.1109/TCAD.2022.3147444

II. Preliminaries

A. Stuck-At Fault

A test pattern for a stuck-at fault at a node is an input pattern that can distinguish the fault-free and the faulty circuits. When

TABLE I
RELATIONSHIPS OF FUNCTIONALITIES BETWEEN 2-INPUT AND GATE AND OTHER LOGIC GATES. X REPRESENTS DC

a b	AND	XNOR	b	a	0
0 0	0	X	0	0	0
0 1	0	0	X	0	0
1 0	0	0	0	X	0
1 1	1	1	1	1	X

such a test pattern exists, the fault is testable. Otherwise, this faulty wire or gate can be replaced with the corresponding faulty value. The mandatory assignments (MAs) are the unique values assigned to some nodes for the existence of a test pattern. The dominators for a node n are a set of nodes that all paths from n to any PO must pass through. A side input to a dominator d of node n is one of d*'s* inputs that is not in the transitive fanout (TFO) cone of n. To find a test pattern, the value given to the faulty node n must activate the fault-effect. Next, the fault-effect has to be propagated to at least one PO. Hence, any side input to any dominator of n has to be assigned a noncontrolling value of the corresponding dominator. After collecting the MAs from the fault activation and fault propagation, we can acquire more MAs by logic implications.

III. PROPOSED DC-BASED APPROACH

In this section, we present the proposed DC-based approach consisting of two techniques. The first one exploits SDCs and ODCs to transform AND gates into XNOR gates in the network represented in XAG. The other technique minimizes the number of AND gates in the XAG by merging AND gates or transforming AND gate into other logic considering ODCs.

A. SDCs & ODCs

Theorem 1: Given a 2-input AND gate in an XAG, when any input pattern of the AND gate is a don't care (DC), the multiplicative complexity of the XAG can be reduced by 1.

Proof: According to the truth table in Table I, the functionalities of AND gate and XNOR gate differ only at the input pattern of $(a, b) = (0, 0)$. Hence, when the input pattern $(a, b) = (0, 0)$ is a DC, the AND gate can be transformed into an XNOR gate. Other three conditions are similar. Thus, when any input pattern of the AND gate is a DC, the AND gate can be transformed into other logic, which reduces the multiplicative complexity of XAG by 1. ■

According to Theorem 1, when any input pattern of an AND gate is a DC, the multiplicative complexity of XAG can be reduced by 1. However, based on our preliminary experiments, the occurrence of (0, 0) being a DC is more common. To balance the quality and CPU time, we only examine whether (0, 0) is a DC in this technique.

In this work, we formulate the process of checking DC as a SAT problem. We can use the Tseytin transformation [17] to transform nodes into its CNF. The nodes transformed into CNF are collected from a region with a moderate size for avoiding large computational overheads. Mishchenko and Brayton [12] proposed an algorithm considering fanout-reconvergent paths to build an $m \times n$ basic window for a target node t, where m and n represent the extended levels to t*'s* TFI and t*'s* TFO, respectively.

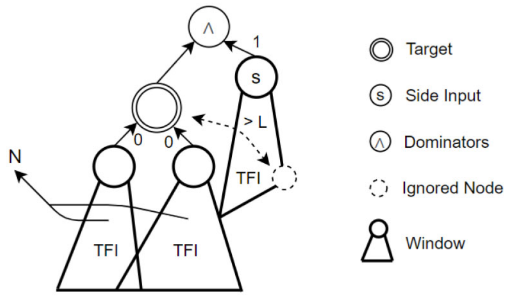

Fig. 1. Demonstration of the enhanced window for the target node.

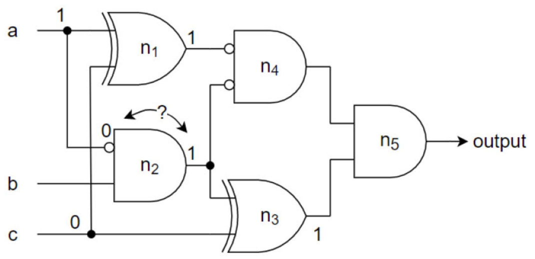

Fig. 2. Example of using DCs for minimizing the number of AND gates.

Furthermore, we enlarge the window to elevate the effect of SDCs and ODCs. Fig. 1 sketches an enhanced window for the target node. For SDCs, to balance the efficiency and effectiveness, we collect at most N nodes in t*'s* TFI cone. For ODCs, we collect side inputs of all the dominators of t and nodes in their TFI cone that are away within L levels with respect to t. That is, a node that is far from t will be excluded. These collected nodes form an enhanced window. The enhanced window and the basic window form a combined window, and C_{window} denotes a set of clauses transformed from the combined window. Next, the input pattern of the target AND node is forced to be (0, 0). C_{input} denotes the clauses with respect to this assignment. Finally, each side input to the corresponding dominator has to be assigned a noncontrolling value, and the corresponding clauses are denoted as C_{side_input}. The conjunction of C_{window}, C_{input}, and C_{side_input} is shown as

$$C_{\text{window}} \wedge C_{\text{input}} \wedge C_{side\_input}. \quad (2)$$

When (2) is unsatisfiable after running a SAT solver, the input pattern (0, 0) is a DC to the target node. As a result, the target node can be transformed into an XNOR gate.

For instance, consider a target node n_4 in a small circuit as shown in Fig. 2. C_{window} is transformed by the whole network in this example. C_{input} is $(n_1) \wedge (n_2)$, which forces n_1 and n_2 to be 1 for producing an input pattern of (0, 0) to n_4. C_{side_input} is (n_3), which forces the side input n_3 of the dominator n_5 to be the noncontrolling value 1. When $n_3 = 1, n_2 = 1$, then $c = 0$. Also, when $n_1 = 1, c = 0$, then $a = 1$. However, $a = 1$ forces $n_2 = 0$, which contradicts the premise about $n_2 = 1$. As a result, (2) is unsatisfiable for this example; i.e., we can transform n_4 into an XNOR gate without changing the functionality of this network. The pseudocode of this technique is shown in Algorithm 1.

B. Node Merging & Node Addition and Removal

In this section, we introduce the node merging (NM) [5], [6] and node addition and removal (NAR) [7], [8] algorithms,

Algorithm 1: SDCs_&_ODCs(Circuit C)

```
foreach node t in C do
    if t is an AND gate then
        combined_window ← Build_Window(t);
        $C_{window}$ ← Build_CNF(combined_window);
        cnf ← $C_{window} \wedge C_{input} \wedge C_{side\_input}$;
        result ← SAT_Solver(cnf);
        if result==UNSAT then
            Transform t into an XNOR gate;
return;
```

which were proposed for reducing the node count in the circuits. We also clarify the differences when applying NM and NAR algorithms in our approach.

Condition 1 [5], [6]: Let f represent an error of replacing a target node n_t with its substitute node n_s. If $n_s = 1$ and $n_s = 0$ are MAs for the stuck-at 0 fault and the stuck-at 1 fault on n_t, respectively, f is undetectable.

Proof: Already demonstrated in [5] and [6]. ■

Condition 1 states a sufficient condition for safely merging a target node n_t with its substitute node n_s. This is because the error of merging n_t with n_s is undetectable under this condition.

Condition 2 [7], [8]: If both $n_{f1} = 1$ and $n_{f2} = 1$ are MAs for the stuck-at 0 fault test on n_t, $n_a = 1$ is an MA for the same test as well.

Condition 3 [7], [8]: If $n_{f2} = 0$ is a value assignment in imp(($n_{f1} = 1$) ∪ MAs(n_t = sa1)), $n_a = 0$ is an MA for the stuck-at 1 fault test on n_t.

Proof: Already demonstrated in [7] and [8]. ■

Chen and Wang [7] and [8] proposed an NAR algorithm, which extended Condition 1 to Conditions 2 and 3 to avoid n_s being limited to existing nodes in the circuit. When Conditions 2 and 3 are held simultaneously, an added AND gate n_a is a valid substitute node to n_t; i.e., we can replace n_t with n_a, where n_{f1} and n_{f2} are two fanin nodes of n_a.

The NM and NAR algorithms proposed in [5]–[8] were applied in AND-inverter graphs (AIGs). Although Conditions 1–3 for the NM and NAR algorithms can be still used under XAG representation, there exist differences in the process of deriving MAs.

In AIGs, all the dominators of the target node can be used for MA derivations. However, in XAGs, a dominator d of the target node is useful only when d is an AND gate. This is because XOR gates have no unique noncontrolling value; i.e., we cannot infer MAs on the side inputs of XOR gates. Thus, we further propose mandatory relations (MRs) for MA derivations in XAGs. We use a simple example to demonstrate the effect of MRs in Fig. 3.

Let us consider a stuck-at 1 fault on n_1. $n_1 = 0$ and $n_2 = 1$ are MAs for the fault activation and propagation as shown in Fig. 3(a). Since n_2 is an XOR gate, $n_2 = 1$ only implies $a \neq b$, but not infers definite values of a and b. We call this relation of $a \neq b$ an MR, and record it for further use if applicable. Next, due to $a \neq b$, $n_4 = 0$ is an MA. Furthermore, by combining $n_1 = 0$ and $a \neq b$, we have $(a, b) = (0, 1)$. Hence, we can conclude $\{n_4 = 0, a = 0, b = 1\} \in$ MAs(n_1 = sa1).

For the first technique described in Section III-A, we use SAT solvers to determine the DCs of AND gates. Here, we exploit the information of MAs to examine DCs by logic implications. Specifically, we assign an input pattern to an AND gate and observe if any conflict occurs on the MAs after logic implications. If there occurs a conflict on the MAs, this input pattern is a DC to the AND node; otherwise, it is not a DC. Note that unlike the first technique, we will check each input pattern of an AND gate about being a DC until we find a DC pattern in the order of $(a, b) = (1, 1), (0, 1), (1, 0), (0, 0)$.

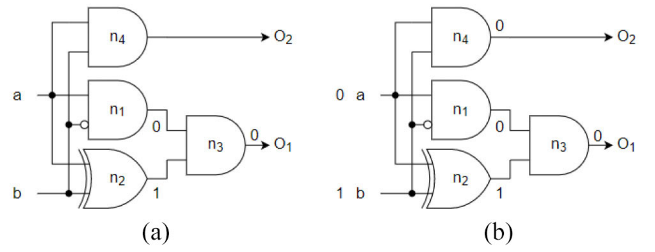

Fig. 3. (a) MAs(n_1=sa1) derived without considering MRs. (b) MAs(n_1 = sa1) derived after considering MRs.

Algorithm 2: NM_&_NAR(Circuit C)

```
foreach node $n_t$ in C do
    if $n_t$ is an AND gate then
        Compute MAs($n_t$ = sa1) and MAs($n_t$ = sa0);
        if there exists $n_s$ then
            Replace $n_t$ with $n_s$;
            continue;
        Check if any input pattern is a DC to $n_t$;
        if there exists a DC input pattern then
            Transform $n_t$ into the corresponding logic;
            continue;
        Compute the size of MFFC of $n_t$;
        if the size of MFFC of $n_t$ > 0 then
            Perform NAR algorithm;
return;
```

Algorithm 2 shows the pseudocode of this technique. Note that the maximum fanout-free cone (MFFC) [15] of a node t represents a set of nodes in t's TFI cone that can be removed when t is removed. When the size of MFFC of n_t is zero, we will not perform NAR algorithm. This is because we at most remove one node, i.e., n_t, as the size of MFFC of n_t is zero. Adding one AND gate n_a and removing another AND gate n_t do not reduce the multiplicative complexity of a circuit.

C. Overall Flow of the Proposed Approach

Our contributions are divided into two parts. In the first part as shown in Fig. 4(a), our approach improves the well-optimized benchmarks achieved by the state-of-the-art [19]. We first apply the SDCs & ODCs technique. Then we apply the NM & NAR technique to obtain further-optimized circuits. For the second part as shown in Fig. 4(b), we propose an enhanced synthesis flow integrating our approach and [19]. In the enhanced flow, we first apply the NM & NAR technique. Then we apply the state-of-the-art method once. Finally, we exploit the SDCs & ODCs technique. The original benchmarks shown in Fig. 4 are the optimized results from [18].

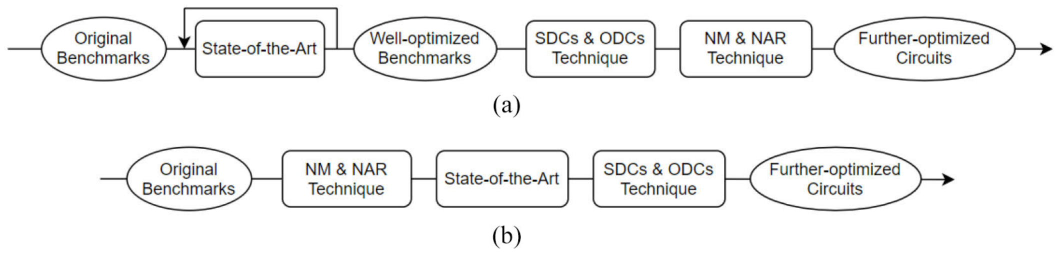

Fig. 4. (a) Flow of improving the well-optimized benchmarks. (b) Proposed enhanced synthesis flow.

TABLE II
COMPARISON OF EXPERIMENTAL RESULTS FOR THE EPFL ARITHMETIC BENCHMARKS

Benchmarks	[19] until no gains		Ours			
	AND	XOR	AND	XOR	Impr. (%)	CPU (s)
Barrel shifter	832	1728	832	1728	0	0.88
Divisor	5291	8678	5132	8830	3.01	22.69
Log2	10913	15923	8773	18059	19.61	312.01
Max	890	1520	872	1531	2.02	7.13
Multiplier	7653	11855	7585	11923	0.89	38.82
Sine	**2603**	**2709**	**1959**	**3350**	**24.74**	**11.10**
Square-root	5381	9260	5217	9419	3.05	68.21
Square	4672	8198	4596	8274	1.63	12.59
Average					6.87	59.18

TABLE III
COMPARISON OF EXPERIMENTAL RESULTS FOR THE CRYPTOGRAPHY BENCHMARKS

Benchmarks	[19] until no gains		Ours			
	AND	XOR	AND	XOR	Impr. (%)	CPU (s)
AES (No Key Expansion)	6800	25124	6800	25124	0	39.41
AES (Key Expansion)	5440	20325	5440	20325	0	30.85
DES (No Key Expansion)	9048	13092	6833	15286	24.48	205.64
DES (Key Expansion)	9205	13136	6915	15406	25.15	222.13
MD5	9367	29729	9367	29729	0	82.78
SHA-1	11515	44358	11483	44389	0.28	152.45
SHA-256	26927	91495	26464	91933	1.72	706.99
32x32-bit Multiplier	1689	3861	1689	3861	0	1.54
Comp. 32-bit Signed LTEQ	92	97	87	102	5.43	0.09
Comp. 32-bit Signed LT	92	95	84	103	8.70	0.10
Comp. 32-bit Unsigned LTEQ	92	97	87	102	5.43	0.09
Comp. 32-bit Unsigned LT	92	95	84	103	8.70	0.09
Average					6.66	120.19

IV. EXPERIMENTAL RESULTS

We implemented the proposed approach in C++ language. The implementation of the state-of-the-art method was from the source code released by [19]. The experiments were conducted on an Intel i9-11900 CPU with 2.5 GHz and 64-GB RAM on a Linux platform (Ubuntu 20.04.02 LTS). The benchmarks are from EPFL arithmetic circuits and cryptography benchmarks, and they are the optimized results in [18] and [19]. The parameters of our approach are set as follows. For the basic window, we use a 2×2 dimension. For the enhanced window, we set N as 300 and L as 200. The SAT solver used in this work is kissat [20].

A. Improvement on the Well-Optimized Benchmarks

In this section, we comprehensively demonstrate the comparison of experimental results in Tables II and III about the improvement on the well-optimized benchmarks.

Table II shows the results for the EPFL arithmetic benchmarks. According to Table II, we can further reduce up to 24.74% of the number of AND gates for *sine* benchmark. The improvement ratio is about 6.87% on average. The average CPU time is less than one minute. Table III shows the results for cryptography benchmarks. Our approach further reduces the number of AND gates for 6.66% $\sim$ 6.87% on average.

B. Proposed Enhanced Synthesis Flow

In this section, we compare the quality and CPU time of the proposed enhanced synthesis flow against the results of [19]. The benchmarks are the optimized results obtained by [18]. The experimental results are shown in Tables IV and V. For example, in Table IV, for *Log2* from the EPFL arithmetic benchmarks, the proposed flow reduces 52.55% AND gates using 930.70 s while the state-of-the-art only reduces 43.85% AND gates using 3412.03 s. According to Table IV, our average improvement is 22.23% while that of the state-of-the-art is only 19.58%. Furthermore, as comparison using the average CPU time, our approach achieved a speedup of 3.32×. For the results of cryptography benchmarks as shown in Table V, the proposed enhanced synthesis flow has the competitive improvements with much less CPU time as compared to the

TABLE IV
RESULTS OF THE PROPOSED ENHANCED FLOW FOR EPFL ARITHMETIC BENCHMARKS

Benchmarks	[18]		Ours				[19] until no gains			
	AND	XOR	AND	XOR	Impr. (%)	CPU (s)	AND	XOR	Impr. (%)	CPU (s)
Barrel shifter	832	1728	832	1728	0	4.45	832	1728	0	3.42
Divisor	6060	8994	5321	8933	12.19	99.74	5291	8678	12.69	545.99
Log2	**19436**	**9371**	**9226**	**17859**	**52.55**	**930.70**	**10913**	**15923**	**43.85**	**3412.03**
Max	931	1479	872	1538	6.34	16.27	890	1520	4.40	7.37
Multiplier	11940	8614	7632	11910	36.08	199.80	7653	11855	35.90	494.25
Sine	4075	1770	2084	3291	48.86	58.15	2603	2709	36.12	207.23
Square-root	6244	9640	5445	9397	12.80	207.33	5381	9260	13.82	347.79
Square	5181	8084	4714	8237	9.01	54.55	4672	8198	9.82	203.36
Average					22.23	196.37			19.58	652.68
Ratio						1				3.32

TABLE V
RESULTS OF THE PROPOSED ENHANCED FLOW FOR CRYPTOGRAPHY BENCHMARKS

Benchmarks	[18]		Ours				[19] until no gains			
	AND	XOR	AND	XOR	Impr. (%)	CPU (s)	AND	XOR	Impr. (%)	CPU (s)
AES (No Key Expansion)	6800	25124	6800	25124	0	199.05	6800	15124	0	162.10
AES (Key Expansion)	5440	20325	5440	20325	0	141.46	5440	20325	0	113.94
DES (No Key Expansion)	15093	11105	6884	15427	54.39	758.14	9048	13092	40.05	1264.90
DES (Key Expansion)	15126	11263	6899	15514	54.39	787.71	9205	13136	39.14	2008.20
MD5	9381	30325	9374	29740	0.10	340.98	9367	29729	0.15	929.87
SHA-1	11820	44311	11569	44344	2.12	700.63	11515	44358	2.58	3204.10
SHA-256	30201	91278	27887	91630	7.66	4034.50	26927	91495	10.84	62185.00
32x32-bit Multiplier	4107	2473	2590	3868	36.94	24.46	1689	3861	58.90	36.74
Comp. 32-bit Signed LTEQ	114	89	96	98	15.79	2.22	92	97	19.30	7.78
Comp. 32-bit Signed LT	108	116	91	102	15.74	2.98	92	95	14.80	9.72
Comp. 32-bit Unsigned LTEQ	114	89	96	98	15.79	2.29	92	97	19.30	7.77
Comp. 32-bit Unsigned LT	108	116	91	102	15.74	2.98	92	95	14.80	9.73
Average					18.22	583.12			18.32	5828.32
Ratio						1				9.99

state-of-the-art. The speedup is almost 10× when considering average CPU time.

V. CONCLUSION

In this work, we proposed a DC-based approach to further reduce the multiplicative complexity of the well-optimized benchmarks. Furthermore, we propose an enhanced synthesis flow integrating our approach and the state-of-the-art. The proposed enhanced synthesis flow achieves competitive improvements with much less CPU time as compared to the flow in the state-of-the-art.

REFERENCES

[1] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *Proc. Eurocrypt*, 2015, pp. 430–454.

[2] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 6, pp. 818–830, Jun. 2013.

[3] J. Boyar, P. Matthews, and R. Peralta, "Logic minimization techniques with applications to cryptology," *J. Cryptol.*, vol. 26, no. 2, pp. 280–312, 2013.

[4] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 1, pp. 1–12, Jan. 1994.

[5] Y.-C. Chen and C.-Y. Wang, "Fast detection of node mergers using logic implications," in *Proc. ICCAD*, 2009, pp. 785–788.

[6] Y.-C. Chen and C.-Y. Wang, "Fast node merging with don't cares using logic implications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 11, pp. 1827–1832, Nov. 2010.

[7] Y.-C. Chen and C.-Y. Wang, "Node addition and removal in the presence of don't cares," in *Proc. DAC*, 2010, pp. 505–510.

[8] Y.-C. Chen and C.-Y. Wang, "Logic restructuring using node addition and removal," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 2, pp. 260–270, Feb. 2012.

[9] M. Chase *et al.*, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *Proc. CCS*, 2017, pp. 1825–1842.

[10] S. Jang, K. Chung, A. Mishchenko, and R. Brayton, "A power optimization toolbox for logic synthesis and mapping," in *Proc. IWLS*, 2009, pp. 1–8.

[11] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Proc. ICALP*, 2008, pp. 486–498.

[12] A. Mishchenko and R. K. Brayton, "SAT-based complete don't-care computation for network optimization," in *Proc. DATE*, 2005, pp. 412–417.

[13] M. Miller and M. Soeken, "An algorithm for linear, affine and spectral classification of Boolean functions," in *Proc. Int. Workshop Boolean Problems*, 2018, pp. 237–254.

[14] G. Meuli, M. Soeken, E. Campbell, M. Roetteler, and G. De. Micheli, "The role of multiplicative complexity in compiling low T-count oracle circuits," in *Proc. ICCAD*, 2019, pp. 1–8.

[15] H. Riener, W. Haaswijk, A. Mishchenko, G. De Micheli, and M. Soeken, "On-the-fly and DAG-aware: Rewriting Boolean networks with exact synthesis," in *Proc. DATE*, 2019, pp. 1649–1654.

[16] M. Soeken, E. Testa, and D. M. Miller, "A hybrid method for spectral translation equivalent Boolean functions," in *Proc. PACRIM*, 2019, pp. 1–6.

[17] G. S. Tseytin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II* (Seminars in Mathematics), A. O. Slisenko Ed. New York, NY, USA: Consultants Bureau, 1970, pp. 115–125.

[18] E. Testa, M. Soeken, L. Amaru, and G. De. Micheli, "Reducing the multiplicative complexity in logic networks for cryptography and security applications," in *Proc. DAC*, 2019, pp. 1–6.

[19] E. Testa, M. Soeken, H. Riener, L. Amaru, and G. De. Micheli, "A logic synthesis toolbox for reducing the multiplicative complexity in logic networks," in *Proc. DATE*, 2020, pp. 568–573.

[20] "Kissat." [Online]. Available: http://fmv.jku.at/kissat/ (Accessed: Mar. 2021).