Exact Synthesis of Boolean Functions in Majority-of-five Forms

Zhufei Chu*, Winston Haaswijk †, Mathias Soeken†, Yinshui Xia*, Lunyao Wang* and Giovanni De Micheli†
*Faculty of Electrical Engineering & Computer Science, Ningbo University, Ningbo, China
†Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
Email: chuzhufei@nbu.edu.cn

***Abstract*—Recent studies show that majority-based logic synthesis is beneficial for both traditional and nanotechnology digital designs. However, most of the existing synthesis algorithms for majority logic generate majority-of-three (M_3) networks. The optimization opportunity for majority logic by using an arbitrary number of odd inputs still requires a large research effort. In this paper, we present an exact synthesis approach for computing Boolean functions in majority-of-five (M_5) forms with a minimum number of operations using Boolean satisfiability. By exploiting the symmetry properties of majority operators, we make use of symbolic encoding method to represent the node functionality and to reduce the number of variables. Moreover, we represent the M_5 forms by M_5-inverter graphs (M_5IGs) for manipulation, which is an extension of majority-inverter graphs (MIGs). The experimental results on EPFL benchmark suites indicate the proposed method achieves 10.4% improvement on size and 11.4% on depth compared to the state-of-the-art exact synthesis method.**

I. Introduction

Logic synthesis plays an essential role within *computer-aided design* (CAD) systems for digital circuits. New data structures and algorithms for logic synthesis are motivated by both the search for faster circuits in CMOS and the emergence of nanotechnologies (e.g., *Quantum-dot Cellular Automata* QCA [1]) where *majority logic* plays a key role. This led to a renewed interest on majority synthesis and optimization [2], [3], [4], [5], yielding competitive results in CMOS ASICs, *Field Programmable Gate Arrays* (FPGAs) besides emerging technologies.

Most of the existing synthesis algorithms for majority logic exploit majority-of-three (M_3) networks, that can be generalized to the majority of an arbitrary odd number n of inputs. Although theoretical results were presented in [6], efficient optimization methods are still missing. Nevertheless, QCA implementations of adders show that majority-of-five (M_5)-based designs has superior performance and area [7] as compared to M_3-based designs.

In this paper, we synthesize Boolean functions in M_5 forms using a new exact synthesis technique, i.e., with guaranteed minimality properties. First, we exploit the identities of M_5 operators and show how to map them into the commonly-used M_3, AND, and OR operations. Then, we extend the state-of-the-art exact synthesis algorithms to support M_5 operations. In particular, the number of encoding variables (critical for effective optimization) is reduced by exploiting the symmetry properties of M_5. We conduct experiments on 4-variable Boolean functions by computing its optimal M_5IG logic networks. The results show that the upper bound on the number of M_5IG gates for representing 4-variable functions is 5 with a depth of 5 levels. The proposed method is used to synthesize arithmetic benchmark suites [8]. On average, our method achieves 10.4% improvement on size and 11.4% on depth as compared to the exact synthesis using *Majority-Inverter Graphs* (MIGs) [5] as the underlying data structure. The size and depth improvement are promising for an efficient nanotechnology circuit design with better figures of merit of area and performance.

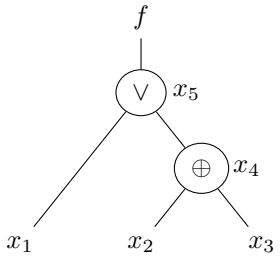

Fig. 1. The Boolean network of Example 1, $x_4 = x_2 \oplus x_3$ and $x_5 = x_1 \vee x_4$.

II. Background

A. Boolean Functions and Networks

The functions considered in this paper are completely specified Boolean functions $f : \mathbb{B}^n \to \mathbb{B}$, and $\mathbb{B} \in \{0, 1\}$. Given a set of Boolean variables $X = \{x_1, \ldots, x_n\}$, a function $f(X)$ can be represented by its truth table which is a 2^n size bitstring $f = (b_{2^n-1} \ldots b_0)$, where b_i, $i \in [0, 2^n - 1]$ is the bit position in the truth table.

Example 1. *The truth table of the function $x_1 \vee (x_2 \oplus x_3)$ can be represented in either $f = (1011\,1110)_2$ or 0xbe in hexadecimal form.*

A Boolean network is a directed acyclic graph (DAG) with nodes corresponding to Boolean functions and edges corresponding to wires connecting the nodes [9]. Mathematically, given a function of n inputs $x_1, \ldots, x_n$, a Boolean network is a sequence of gates $(x_{n+1}, \ldots, x_{n+r})$ with

$$x_i = x_{j(i)} \circ_i x_{k(i)}, \qquad \text{for } n+1 \leq i \leq n+r \tag{1}$$

That means the two inputs of each gate i are previous gates or inputs with $j(i) < k(i) < i$ using $\circ_i$, which is one of the 16 binary operations [10]. The last gate x_{n+r} is the network's output for single-output functions, while each gate could potentially be an output for multi-output networks. The Boolean network of Example 1 is shown in Fig. 1. A Boolean function f is called *normal* if $f(0, \ldots, 0) = 0$. A Boolean

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

network represents a normal function if all of its gate functions are normal.

B. Majority-Based Logic Synthesis

The M_3 function f over three Boolean variables a, b, and c is denoted by $f = \langle abc \rangle$, which can be expressed in both disjunctive and conjunctive normal forms (CNFs).

$$f = ab \vee ac \vee bc = (a \vee b)(a \vee c)(b \vee c) \tag{2}$$

By setting any variable to constant 0 and 1, one can obtain the conjunction and disjunction of the other two variables, respectively. The MIGs are logic representations that use only M_3 and inverters as basic primitives. The axiomatic system for the MIG Boolean algebra makes MIG-based representations extremely competitive at logic rewriting.

In terms of M_5 logic function over five Boolean variables a, b, c, d, e,

$$\begin{aligned}\langle abcde \rangle =& abc \vee abd \vee abe \vee acd \vee ace \vee \\ & ade \vee bcd \vee bce \vee bde \vee cde\end{aligned} \tag{3}$$

However, it can be expressed in terms of M_3, which resulted in an optimal depth expression using M_3-based exact synthesis [2].

$$\langle abcde \rangle = \langle a \langle bcd \rangle \langle \langle abc \rangle de \rangle \rangle \tag{4}$$

The optimization opportunities arise by applying the identity from right to left in a MIG, in which the depth can be reduced by 2 and the number of nodes can be reduced by 3 in the respective subcircuit.

III. SAT Based Exact Synthesis

In this Section, we first demonstrate the SAT formulation proposed by Knuth to find an area-optimal normal network. Then we propose our encoding method for M_5 operators.

A. Knuth's Algorithm

Knuth's algorithm aims to find a normal Boolean network, or Boolean chains, using 2-input gates. It was inspired by the work of Kojevnikov [11] et al. and Éen [12]. Recently, the SAT formulation was extended for combinational delay optimization [13] and logic synthesis applications with complex constraints [14].

1) Variables: For $1 \leq h \leq m$, $n < i \leq n + r$, and $0 < t < 2^n$, the variables used in the SAT formulation are defined in the following:

$$\begin{aligned} x_{it}: &\quad t^{th} \text{ bit of } x_i\text{'s truth table} \\ g_{hi}: &\quad [g_h = x_i] \\ s_{ijk}: &\quad [x_i = x_i \circ_i x_k] \text{ for } 1 \leq j < k < i \\ f_{ipq}: &\quad \circ_i(p, q) \text{ for } 0 \leq p, q \leq 1, p + q > 0 \end{aligned} \tag{5}$$

If g_{hi} is true, it means function g_h is represented by gate x_i. The variable s_{ijk} is a selection variable, which evaluates to true if the two inputs of gate x_i are x_j and x_k. Finally, the variable f_{ipq} is true, if the operation of gate x_i is true for input assignment (p, q). Note that the method works for normal Boolean functions. If a function is not normal, we invert the root gate to generate a inverted function for preprocessing. Because the function is normal, which inherently makes each gate maps $(0, 0) \mapsto 0$, we discard x_{i0} and f_{i00} for all i. We refer the reader to [13] for a comprehensive example to show the variables assignment. The defined variables are then constrained by a set of clauses to ensure the network realizes the correct functions.

2) Clauses: Intuitively, if gate x_i has two inputs x_j and x_k, and the t^{th} bit of x_i, x_j, and x_k are a, b, and c, respectively, then the gate x_i must operate as $b \circ_i c = a$. Thus the main clauses to represent the operation constraints can be written as:

$$((s_{ijk} \wedge (x_{it} \oplus \bar{a}) \wedge (x_{jt} \oplus \bar{b}) \wedge (x_{kt} \oplus \bar{c})) \mapsto (f_{ibc} \oplus \bar{a}) \tag{6}$$

Note that a, b, and c are constants which are used to set the proper variable polarities. It can be rewritten as CNFs, that is

$$(\bar{s}_{ijk} \vee (x_{it} \oplus a) \vee (x_{jt} \oplus b) \vee (x_{kt} \oplus c)) \vee (f_{ibc} \oplus \bar{a})) \tag{7}$$

Let $(t_1, \ldots, t_n)_2$ be the binary encoding of t, then the clauses

$$(\bar{g}_{hi} \vee (\bar{x}_{it} \oplus g_h(t_1, \ldots, t_n))) \tag{8}$$

constrain the output values to the gates they point to. Moreover, the constraints $\bigvee_{i=n+1}^{n+r} g_{hi}$ ensure that each output is realized by the network and the constraints $\bigvee_{k=2}^{i-1} \bigvee_{j=1}^{k-1} s_{ijk}$ ensure that each gate has exactly two inputs.

The above-mentioned clauses are essential to make the algorithm work. However, additional constraints can help to reduce the search space for the SAT solver [10]. Especially, a recent work that using DAG topologies to constraint the shape of the network is promising in runtime [3].

B. Identities of M_5 Operators

We consider using M_5 as logic primitives for the synthesis of Boolean functions. In our case, we make use of *M_5-Inverter Graphs* (M_5IG) as the underlying data structure for exact synthesis.

Theorem 1. *The M_5 operator can be reduced to M_3 function if 1) there exists two pair of duplicated inputs, or 2) the two inputs biased to constant inputs 0 and 1.*

Proof. Without loss of generality, we assume the two duplicated inputs are a and b, then

$$\langle aabbc \rangle = \langle abc \rangle \tag{9}$$

One can obtain the right hand side expression by expanding and simplifying of the left hand side function defined in Equation (3). Also, we assume the five inputs of M_5 contains two constant inputs 0 and 1, while the other three inputs are a, b, and c, then

$$\langle 01abc \rangle = \langle abc \rangle \tag{10}$$

The expansion and simplification processes are similar with the proof of Equation (9). □

Due to symmetries, one can obtain more identities as follows:

$$\langle aabbc \rangle = \langle aabcc \rangle = \langle abbcc \rangle = \langle 01abc \rangle \tag{11}$$

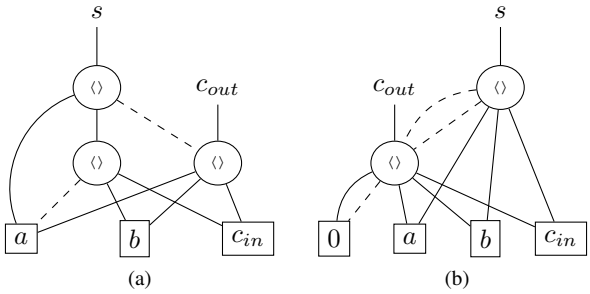

Fig. 2. A full adder logic network represented by (a) MIG and (b) M_5IG, where the dashed lines indicate the complemented edges. Note that the constant zero input is not shown in (a) as it is not used.

Moreover, as AND and OR operators can be obtained by setting one input of M_3 to a constant, in terms of M_5 operator, they can also behave as AND and OR by

$$a \wedge b = \overbrace{\langle 0ab\rangle}^{\text{Equation (9)}} = \langle 00aab\rangle = \langle 010ab\rangle \tag{12}$$

$$a \vee b = \underbrace{\langle 1ab\rangle}_{\text{Equation (10)}} = \langle 11aab\rangle = \langle 011ab\rangle \tag{13}$$

Theorem 2. *$M_5IG \supset MIG$*

Proof. M_5IG is an extension of MIG, which is a homogeneous logic network with an indegree equal to 5 and each node representing the M_5 function. In both M_5IGs and MIGs, the complemented edges represent inverters. A MIG node is always a special case of an M_5IG node, as can be obtained from Theorem 1. On the other hand, an M_5IG node is never a special case of a MIG node, because of the functionality of the M_5 cannot be uniquely realized by M_3. □

Corollary 1. *$M_5IG \supset$* And-Inverter Graph *(AIG) and M_5IG is an universal representation form.*

Proof. M_5IG $\supset$ MIG $\supset$ AIG, where both MIG and AIG are universal representation forms [5], [15]. □

Fig. 2 depicts MIG and M_5IG logic networks for a full adder, which can be expressed by

$$s = a \oplus b \oplus c_{in} \text{ and } c_{out} = \langle abc_{in}\rangle \tag{14}$$

Both of the logic networks are optimal representations using the given primitives [6].

A proper set of manipulation tools are essential to handle M_5IG to automatically reach compact representations. Although an axiomatization system for majority-n logic was presented, how to obtain an effective initial M_5IG representation has not been addressed. Obviously, exact synthesis using M_5IG as the underlying data structure can reach a more compact logic network than the method using one-to-one replacement of MIG nodes by M_5IG nodes.

C. Encoding of M_5 Constraints

The variables to encode the truth table and the output gate are the same with Knuth's method. As we constrain the node functionality to be M_5, which has five inputs instead of two, the selection variable has extended to be s_{iJ} which means the node i has five inputs from the set $J = 0, a, b, c, d, e$. To cover all possibilities of 5-inputs, we allow both constant and duplicated inputs, e.g., both s_{i00abc} and s_{iaabbc} are valid representations. Given the inputs set J, to construct input combinations, we consider following cases of the inputs to cover the ordinary 2-5 individual inputs:

- five inputs: $\langle abcde\rangle$ for M_5 operation, there is just one case, $\binom{5}{5} = 1$.
- four inputs: considering one pair of duplicated inputs of the forms $\langle aabcd\rangle, \ldots, \langle bcdee\rangle$, thus there is totally $4 \times \binom{5}{4}$ cases; also in terms of one constant input, there are totally $\binom{5}{4}$ cases of the forms $\langle 0abcd\rangle, \ldots, \langle 0bcde\rangle$.
- three inputs: considering two constant inputs to build M_3 operation, for the forms of $\langle 00abc\rangle, \ldots, \langle 00cde\rangle$, there are totally $\binom{5}{3}$ cases.
- two inputs: considering two-operands AND and OR operations, for the forms of $\langle 00aab\rangle, \ldots, \langle 00dde\rangle$, there are totally $\binom{5}{2}$ cases.

Therefore, assume that we are given 5 non-constant inputs and a constant 0 input, the number of required steps to compute the function is 1, then we need

$$\binom{5}{5} + 5 \times \binom{5}{4} + \binom{5}{3} + \binom{5}{2} = 46 \tag{15}$$

selection variables. Generally, suppose we are given n_{in} non-constant inputs and a constant 0 input, the number of required steps to compute the function is n_{req}, and $n_t = n_{in} + n_{req} - 1$, then we need

$$\binom{n_t}{5} + 5 \times \binom{n_t}{4} + \binom{n_t}{3} + \binom{n_t}{2} \tag{16}$$

selection variables to encode SAT formulation.

Since the majority function is self-dual, which is $\overline{\langle abcde\rangle} = \langle \bar{a}\bar{b}\bar{c}\bar{d}\bar{e}\rangle$, we only consider the following 16 cases, while the other 16 cases can be obtained by inverting the function outputs.

$\langle abcde\rangle$	$\langle \bar{a}bcde\rangle$	$\langle a\bar{b}cde\rangle$	$\langle ab\bar{c}de\rangle$
$\langle abcd\bar{e}\rangle$	$\langle abcd\bar{e}\rangle$	$\langle \bar{a}\bar{b}cde\rangle$	$\langle \bar{a}b\bar{c}de\rangle$
$\langle \bar{a}bcd\bar{e}\rangle$	$\langle \bar{a}bcd\bar{e}\rangle$	$\langle a\bar{b}\bar{c}de\rangle$	$\langle a\bar{b}cd\bar{e}\rangle$
$\langle a\bar{b}cd\bar{e}\rangle$	$\langle ab\bar{c}d\bar{e}\rangle$	$\langle ab\bar{c}d\bar{e}\rangle$	$\langle abcd\bar{e}\rangle$

Knuth's method use variables f_{ipq} to indicate the operations for gate x_i under the input assignment (p, q), thus the f_{ipq} allow for a representation of all $2^{2^2} = 16$ normal 2-input functions. In our scenario, this number will dramatically increase to 2^{2^5} normal 5-input functions. Therefore, we use symbolic encoding method to represent all 16 M_5 functions. The operation variable for step r is encoded as $o_{r1}, \ldots, o_{r16}$, we need add additional two clauses.

- Clause $\bigvee_{w=1}^{16} o_{rw}$ ensure that each step should realize at least one of the 16 operations.

- For each selection variable and all input combinations from (00000) to (11111), we check the output of all the 16 operations to add consistency constraints to ensure the operations compute the correct functions. For example, suppose the selection variable s_{iabcde}, and the input combination is (10000), then we can check $o_{r1} = \langle abcde \rangle$ outputs 0, while $o_{r16} = \langle abcd\bar{d}\bar{e} \rangle$ outputs 1. One can verify that the output offset is $\{o_{r1}, \ldots, o_{r10}\}$ and the onset is $\{o_{r11}, \ldots, o_{r16}\}$. Thus the clause is added as follows.

$$\begin{aligned}
&\bar{s}_{iabcde} \vee \bar{x}_{it} \vee o_{r11} \vee \ldots \vee o_{r16} \\
&\bar{s}_{iabcde} \vee \bar{x}_{it} \vee \bar{o}_{r1} \\
&\ldots \\
&\bar{s}_{iabcde} \vee \bar{x}_{it} \vee \bar{o}_{r10} \\
&\bar{s}_{iabcde} \vee x_{it} \vee o_{r1} \vee \ldots \vee o_{r10} \\
&\bar{s}_{iabcde} \vee x_{it} \vee \bar{o}_{r11} \\
&\ldots \\
&\bar{s}_{iabcde} \vee x_{it} \vee \bar{o}_{r16}
\end{aligned} \tag{17}$$

Therefore, the Knuth's method can be extended to solve exact synthesis of Boolean functions using M_5 operators.

Given a Boolean function, we start the exact synthesis algorithm by trying to find a solution using $r = 1$ gate. If one solution is found, it returns an M_5IG; otherwise, the algorithm increases the number of gates r to restart encoding and solving until the upper bound is reached, which ensures the algorithm find the logic network with an optimal number of gates.

IV. Experiments

The proposed exact synthesis method is implemented in C++ based on EPFL open source logic synthesis libraries [16]. All experiments were conducted on an Intel® Xeon® CPU E5-2650 v4 @ 2.20GHz. The results are verified by simulating the truth tables to ensure correctness.

A. Evaluations on 4-variable Boolean Functions

Two Boolean functions f and g are *NPN-equivalent* if f can be obtained from g by **n**egating inputs, **p**ermuting inputs, or **n**egating the output. All 4-input Boolean functions can be classified into only 222 NPN representatives. We implemented our exact synthesis algorithm to all 222 NPN classes. The results show the most expensive function $f = a \oplus b \oplus c \oplus d$ requires 5 M_5IG nodes with a depth of 5 levels instead of 7 MIG nodes with a depth of 6 levels [2]. Therefore, we can obtain advantages on both size and depth of the logic network. The expressions of f are shown as follows.

$$\begin{aligned}
x_1 &= \langle \bar{0}0a\bar{b}c \rangle & x_2 &= \langle \bar{a}b\bar{c}x_1x_1 \rangle \\
x_3 &= \langle 00\bar{d}\bar{d}x_2 \rangle & x_4 &= \langle b\bar{c}x_1x_3x_3 \rangle \\
x_5 &= \langle \bar{0}d\bar{x}_2x_3x_4 \rangle & f &= x_5
\end{aligned}$$

The computation time for all these functions is around 8 hours, which indicate 2 minutes are required on average for each function. However, the computation time can be improved using modern SAT encoding techniques such as counterexample guided abstraction refinement. The prior knowledge about the Boolean functions structures by decomposition is also helpful for SAT solving.

TABLE I
Comparing MIG and M5IG size/depth optimization

Benchmark	I/O	MIG [2]		M_5IG	
		Size	Depth	Size	Depth
Adder	256/129	512	130	**386**	**129**
Barrel shifter	135/128	3238	14	**2496**	14
Divisor	128/128	44331	4381	47147	**4231**
Hypotenuse	256/128	160678	9518	**141850**	**9334**
Log2	32/32	27645	383	**22314**	**338**
Max	512/130	2535	294	**2302**	**237**
Multiplier	128/128	22720	188	**19362**	**186**
Sine	24/25	4768	169	**3822**	**157**
Square-root	128/64	19746	6043	**16972**	**4097**
Square	64/128	15670	156	**13855**	**129**
Average:		30184	2128	**27051 (10.4%↓)**	**1885 (11.4%↓)**
Geomean:		2201		**1929 (12.4%↓)**	

B. Evaluations on EPFL Combinational Benchmarks

To apply our method to large circuits, we conduct experiments on 10 EPFL arithmetic combinational benchmarks. We first apply LUT mapping on all circuits to map the network into k-LUTs. Each LUT represents a k-variable Boolean function, which serves as the input of our exact synthesis. Since all 4-variable functions only have 222 NPN classes and the optimal M_5IG logic networks are precomputed, we set $k = 4$ to replace each LUT with optimum networks and finally merge them together to construct an optimized, functionally equivalent logic network.

The results are shown in Table I, in which the information of the benchmark name, primary inputs/outputs (I/O) are listed in the first two columns. For comparison, we compare the method with exact synthesis using MIG as the underlying data structures [2]. On average, our method can achieve 10.4% improvement on size while 11.4% on depth. After computing the geometric mean over the sizes and depths, the proposed method performs 12.4% better than MIG. In terms of size, 9 out of 10 benchmarks are optimized except circuit `Divisor`. The main reason is our method require at least 5-variable inputs, for these function less than 5, we first extend them to 5-inputs. Thus, if the LUT mapping generates too many functions with a small number of variables, it may result in performance deterioration. In terms of depth, 9 out of 10 benchmarks achieve improvement while circuit `Barrel shifter` got exactly the same depth.

V. Conclusion

Majority-based logic synthesis is promising for both traditional and emerging digital circuit designs. Most current synthesis algorithm using M_3 as logic primitive since it is simple and comprehensively studied. In this paper, we presented an exact synthesis method to represent Boolean functions in M_5 forms. The experimental results on EPFL benchmark suites show that we obtain 10.4% improvement on size and 11.4% on depth compared to the method based on M_3.

Acknowledgment

This research was partly supported by NSF China (61871242) and by Zhejiang Provincial NSF (Y19F040013).

REFERENCES

[1] I. Amlani, A. O. Orlov, G. Toth, G. H. Bernstein, C. S. Lent, and G. L. Snider, "Digital logic gate using quantum-dot cellular automata," *Science*, vol. 284, no. 5412, pp. 289–291, 1999.

[2] M. Soeken, L. G. Amaru, P.-E. Gaillardon, and G. De Micheli, "Exact synthesis of majority-inverter graphs and its applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1842–1855, 2017.

[3] W. Haaswijk, A. Mishchenko, M. Soeken, and G. De Micheli, "SAT based exact synthesis using DAG topology families," in *Desgin Automation Conference (DAC)*, 2018.

[4] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Design Automation Conference (DAC)*, 2014, pp. 1–6.

[5] ——, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Trans on Computer-aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2016.

[6] L. Amaru, P.-E. Gaillardon, A. Chattopadhyay, and G. De Micheli, "A sound and complete axiomatization of majority-n logic," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2889–2895, 2016.

[7] K. Navi, S. Sayedsalehi, R. Farazkish, and M. R. Azghadi, "Five-input majority gate, a new device for quantum-dot cellular automata," *Journal of Computational and Theoretical Nanoscience*, vol. 7, no. 8, pp. 1546–1553, 2010.

[8] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *International Workshop on Logic and Synthesis (IWLS)*, 2015, pp. 57–61.

[9] A. Mishchenko and R. Brayton, "Integrating an AIG package, simulator, and SAT solver," in *International Workshop on Logic and Synthesis (IWLS)*, 2018, pp. 11–16.

[10] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Massachusetts: Addison-Wesley, 2015.

[11] A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, "Finding efficient circuits using SAT-solvers," in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2009, pp. 32–44.

[12] N. Eén, "Practical SAT-a tutorial on applied satisfiability solving," *Slides of invited talk at Formal Methods in Computer-Aided Design (FMCAD)*, 2007.

[13] M. Soeken, G. De Micheli, and A. Mishchenko, "Busy man's synthesis: Combinational delay optimization with SAT," in *Proceedings of the Design, Automation & Test in Europe (DATE)*, 2017, pp. 830–835.

[14] E. Testa, M. Soeken, O. Zografos, F. Catthoor, and G. De Micheli, "Exact synthesis for logic synthesis applications with complex constraints," in *International Workshop on Logic and Synthesis (IWLS)*, 2017, pp. 21–27.

[15] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2010, pp. 24–40.

[16] M. Soeken, H. Riener, W. Haaswijk, and G. De Micheli, "The EPFL logic synthesis libraries," *International Workshop on Logic and Synthesis (IWLS), pre-print available at arXiv:1805.05121*, 2018.