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 (M3) 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 (M5) 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 M5 forms by M5-inverter graphs (M5IGs) 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 (M3) 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 (M5)-based designs have superior performance and area [7] as compared to M3-based designs.

In this paper, we synthesize Boolean functions in M5 forms using a new exact synthesis technique, i.e., with guaranteed minimality properties. First, we exploit the identities of M5 operators and show how to map them into the commonly-used M3, AND, and OR operations. Then, we extend the state-of-the-art exact synthesis algorithms to support M5 operations. In particular, the number of encoding variables (critical for effective optimization) is reduced by exploiting the symmetry properties of M5. We conduct experiments on 4-variable Boolean functions by computing its optimal M5IG logic networks. The results show that the upper bound on the number of M5IG 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.

II. BACKGROUND

A. Boolean Functions and Networks

The functions considered in this paper are completely specified Boolean functions \( f : \mathbb{B}^n \rightarrow \mathbb{B} \), 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 \( x_1 \lor (x_2 \oplus x_3) \) can be represented in either \( f = (10111110)\) or \( 0x\) be 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

\[
x_{i} = x_{j(i)} \circ_i x_{k(i)}, \quad \text{for } n+1 \leq i \leq n+r \tag{1}
\]

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
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).

$$f = ab \lor ac \lor bc = (a \lor b)(a \lor c)(b \lor c) \quad (2)$$

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 $M_3$ logic function over five Boolean variables $a$, $b$, $c$, $d$, $e$,

$$\langle abde \rangle = abc \lor abd \lor abc \lor acd \lor ace \lor ade \lor bcd \lor bce \lor bde \lor cde \quad (3)$$

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

$$\langle abde \rangle = \langle abc \rangle \langle \langle abc \rangle de \rangle \quad (4)$$

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 $M_3$ 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 Een [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:

$$x_{it} : \text{$t^{th}$ bit of $x_i$’s truth table}$$

$$g_h : \langle \bar{g}_h = x_i \rangle$$

$$s_{ijk} : \langle x_i = x_k \otimes x_k \rangle \text{ for } 1 \leq j < k < i \quad (5)$$

$$f_{ipq} : \otimes(p, q) \text{ for } 0 \leq p, q \leq 1, p + q > 0$$

If $g_h$ 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 $(0, 0) \rightarrow 0$, we discard $x_{it}$ and $f_{ipq}$ 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 $b \otimes c = a$. Thus the main clauses to represent the operation constraints can be written as:

$$\langle s_{ijk} \land (x_{it} \oplus \bar{a}) \land (x_{jt} \oplus \bar{b}) \land (x_{kt} \oplus \bar{c}) \rangle \rightarrow (f_{ipq} \oplus \bar{a}) \quad (6)$$

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

$$\langle \bar{s}_{ijk} \lor (x_{it} \oplus a) \lor (x_{jt} \oplus b) \lor (x_{kt} \oplus c) \lor (f_{ipq} \oplus \bar{a}) \rangle \quad (7)$$

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

$$\langle g_{hi} \lor \langle \bar{x}_{it} \oplus g_h(t_1, \ldots, t_n) \rangle \rangle \quad (8)$$

constrain the output values to the gates they point to. Moreover, the constraints $\bigvee_{i=n+1}^{m+n} g_{hi}$ ensure that each output is realized by the network and the constraints $\bigvee_{k=2}^{n+1} \bigvee_{j=1}^{k-1} s_{ijk}$ 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 ($M_5$IG) 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

$$\langle aabb \rangle = \langle abc \rangle \quad (9)$$

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

$$\langle 01abc \rangle = \langle abc \rangle \quad (10)$$

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

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

$$\langle aabb \rangle = \langle aabc \rangle = \langle abbcc \rangle = \langle 01ab \rangle \quad (11)$$
Moreover, as AND and OR operators can be obtained by setting one input of M$_5$ to a constant, in terms of M$_5$ operator, they can also behave as AND and OR by

\[ a \land b = \langle 0ab \rangle = \langle 00aab \rangle = \langle 010ab \rangle \quad (12) \]
\[ a \lor b = \langle 1ab \rangle = \langle 11aab \rangle = \langle 011ab \rangle \quad (13) \]

**Theorem 2.** M$_5$IG $\supset$ MIG

**Proof.** M$_5$IG 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$_5$IGs and MIGs, the complemented edges represent inverters. A MIG node is always a special case of an M$_5$ node, as can be obtained from Theorem 1. On the other hand, an M$_5$IG 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$_5$IG $\supset$ And-Inverter Graph (AIG) and M$_5$IG is an universal representation form.

**Proof.** M$_5$IG $\supset$ MIG $\supset$ AIG, where both MIG and AIG are universal representation forms [5], [15].

Fig. 2 depicts MIG and M$_5$IG logic networks for a full adder, which can be expressed by

\[ s = a \oplus b \oplus c_{in} \text{ and } c_{out} = \langle abc_{in} \rangle \quad (14) \]

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

A proper set of manipulation tools are essential to handle M$_5$IG to automatically reach compact representations. Although an axiomatization system for majority-$n$ logic was presented, how to obtain an effective initial M$_5$IG representation has not been addressed. Obviously, exact synthesis using M$_5$IG 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$_5$IG 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 $s_{i,j}$ which means the node $i$ has five inputs from the set $J = \{a, b, c, d, e\}$. To cover all possibilities of 5-inputs, we allow both constant and duplicated inputs, e.g., both $s_{i0abc}$ and $s_{iaabc}$ 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: $\langle abcede \rangle$ for M$_5$ operation, there is just one case, $\langle \emptyset \rangle = 1$.
- four inputs: considering one pair of duplicated inputs of the forms $\langle aabcd \rangle, \ldots, \langle bede \rangle$, 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 $\langle 0abcd \rangle, \ldots, \langle 0bede \rangle$.
- 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 $\langle 00ab \rangle, \ldots, \langle 00de \rangle$, 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

\[ \binom{5}{1} = 5 \times \binom{5}{4} + \binom{5}{3} + \binom{5}{2} = 46 \quad (15) \]

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

\[ \binom{n_t}{5} = 5 \times \binom{n_t}{4} + \binom{n_t}{3} + \binom{n_t}{2} \quad (16) \]

selection variables to encode SAT formulation.

Since the majority function is self-dual, which is $\langle abcede \rangle = \langle abced \rangle$, we only consider the following 16 cases, while the other 16 cases can be obtained by inverting the function outputs.

\[ \langle abcede \rangle \quad \langle abcede \rangle \quad \langle abcede \rangle \quad \langle abcede \rangle \quad \langle abdec \rangle \quad \langle abdec \rangle \quad \langle abed \rangle \quad \langle abed \rangle \quad \langle abde \rangle \quad \langle abde \rangle \quad \langle abde \rangle \quad \langle abde \rangle \quad \langle abde \rangle \quad \langle abde \rangle \quad \langle abde \rangle \quad \langle abde \rangle \]

Knuth’s method use variables $r_{ipq}$ to indicate the operations for gate $x_i$ under the input assignment $(p, q)$, thus the $r_{ipq}$ allow for a representation of all $2^{2^5} = 16$ 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 $s_{abcdes}$ and the input combination is (1000), then we can check $o_{r1} = \langle abced \rangle$ outputs 0, while $o_{r16} = \langle abcdě \rangle$ 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.

\[
\begin{align*}
\bar{s}_{abcdes} \vee \bar{x}_{it} \vee o_{r11} \vee \ldots \vee o_{r16} \\
\bar{s}_{abcdes} \vee \bar{x}_{it} \vee \bar{o}_{r1} \\
\ldots \\
\bar{s}_{abcdes} \vee \bar{x}_{it} \vee \bar{o}_{r10} \\
\bar{s}_{abcdes} \vee \bar{x}_{it} \vee o_{r1} \vee \ldots \vee o_{r10} \\
\bar{s}_{abcdes} \vee \bar{x}_{it} \vee o_{r11} \\
\ldots \\
\bar{s}_{abcdes} \vee \bar{x}_{it} \vee \bar{o}_{r16}
\end{align*}
\]

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 $M_5$IG; 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 negating inputs, permuting inputs, or negating 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 $f = a \oplus b \ominus c \oplus d$ requires $5 \text{M}_5$IG 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.

\[
\begin{align*}
x_1 = \langle 00abce \rangle & \quad x_2 = \langle \bar{a}b\bar{c}x_1, x_1 \rangle \\
x_3 = \langle 00d\bar{c}x_2 \rangle & \quad x_4 = \langle \bar{b}\bar{c}x_1, x_3 \rangle \\
x_5 = \langle 0d\bar{c}x_2, x_3 \rangle & \quad f = x_5
\end{align*}
\]

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.

\begin{table}[h]
\centering
\caption{Comparing MIG and M5IG size/depth optimization}
\begin{tabular}{|l|l|l|l|}
\hline
Benchmark & I/O & MIG [2] & M5IG \tabularnewline
\hline
& Size & Depth & Size & Depth \tabularnewline
\hline
Adder & 256/129 & 512 & 130 & 386 & 129 \\
Barrel shifter & 135/128 & 3238 & 14 & 2496 & 14 \\
Divisor & 128/128 & 4433 & 4381 & 47147 & 4231 \\
Hypotenuse & 256/128 & 160678 & 9518 & 141850 & 9334 \\
Log2 & 32/32 & 27645 & 383 & 23214 & 338 \\
Max & 512/130 & 2535 & 294 & 2302 & 237 \\
Multiplier & 128/128 & 22720 & 188 & 19362 & 186 \\
Sine & 24/25 & 4768 & 169 & 3832 & 157 \\
Square-root & 128/64 & 19746 & 6043 & 16972 & 4097 \\
Square & 64/128 & 15670 & 156 & 13585 & 129 \\
\hline
Average: & 30184 & 2128 & 27051 & 1885 & (10.4\%↓) & (11.4\%↓) \\
Geomean: & 2201 & 1929 & 141850 & 9334 \\
\hline
\end{tabular}
\end{table}

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 $M_5$IG 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 $M_5$ 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


