# XMG-based Logic Synthesis for Emerging Reconfigurable Nanotechnologies

Shubham Rai* Heinz Riener† Giovanni De Micheli† Akash Kumar*
*CfAED Technische Universität Dresden, Germany
†Integrated Systems Laboratory, EPFL, Lausanne, Switzerland

***Abstract*—Emerging reconfigurable nanotechnologies allow the implementation of self-dual functions with a fewer number of transistors as compared to traditional CMOS technologies. Hence, to achieve better area results for *Reconfigurable Field-Effect Transistors* (RFET)-based circuits, it is imperative that a large portion of a logic representation must be mapped to self-dual logic gates. This, in turn, depends upon how self-duality is preserved in the logic representation during logic optimization and technology mapping. In the present work, we develop logic optimization algorithms using *Xor-Majority Graphs* (XMGs) as a logic representation. Firstly, XMGs are more compact for both unate and binate logic functions as compared to conventional logic representations such as *And-Inverter Graphs* (AIGs) or *Majority-Inverter Graphs* (MIGs). Secondly, the logic primitives used in XMGs, that are, *Majority* and *Xor* gates, can better preserve self-duality as both, the majority-of-three and the odd-input Xor function, are self-dual. Keeping in mind the above two advantages of XMGs, we have implemented Boolean size-optimization methods, a rewriting and a resubstitution algorithm, aiming at better preserving self-duality during logic optimization. We evaluate the proposed algorithms using crafted benchmarks with various levels of self-duality, the EPFL benchmarks, and cryptographic benchmarks. The experimental evaluation shows a direct correlation between the relative numbers of self-dual nodes in the XMGs and the area optimization results after technology mapping for RFET-based circuits. For practical benchmarks with a high self-duality ratio, the XMG-based logic optimisation flow can achieve an area reduction of up to 12% when compared to efficient optimization flows implemented in the academic logic synthesis tool ABC.***

## I. Introduction

The recent development of a modern RISC-V processor made with carbon nanotubes field-effect transistors has brought the focus to efficient yet practical circuits based on emerging post-silicon technologies [29]. The work done in [29] stands out as a stepping stone to solve the ever-increasing problem with CMOS dimension-scaling and the growing skew between cost and performance for CMOS-based circuits [20]. Hence, exploring emerging nano-devices is not just an academic exercise but an imperative demand to meet the requirements of future electronics [25].

Ambipolar nanotechnologies form a class of emerging nanotechnologies which enable both n- and p-type functionality from a single transistor. Various device geometries based on materials like germanium [14], silicon [10, 11], graphene [8] etc. offer ambipolarity. Logic gates based on these emerging reconfigurable nanotechnologies allow more functionality per unit transistor [17]. Recently, it has been demonstrated, that Boolean functions which are *self-dual*, can be implemented efficiently using *Reconfigurable Field-Effect Transistors* (RFETs) [33]. However, in order to enable their efficient integration into commercial electronics, contemporary design automation techniques need to support and utilize their requirements and feature-sets, respectively [16]. In the present work, we explore logic synthesis methods which can specifically take advantage of the self-duality of Boolean functions.

Within design automation, logic synthesis is an integral part which optimizes a logic network in terms of a cost function, typically focusing on the reduction of area or delay. Recently, novel multi-level logic representations such as *Xor-And Graphs* [23] or *Xor-Majoriy Graphs* (XMGs) [22, 28] have been proposed, which enrich conventional *And-Inverter Graphs* (AIGs) [7] and *Majority-Inverter Graphs* (MIGs) [18] with an additional Xor primitive. These new logic representations offer more compactness and enable better runtimes for logic optimization and minimization flows [22, 32].

In the present work, we explore the usage of XMGs in logic optimization methods to achieve area reduction after technology mapping for RFET-based circuits. The two contributions are summarized as follows:

1) Conventional logic representations such as AIGs or MIGs suppress self-duality during logic optimization. We propose an XMG-based logic synthesis flow that allows preserving the self-duality during logic optimization. This flow enables better area reductions after technology mapping for RFET-based circuits in comparison to state-of-the-art logic optimization flows. In an experimental evaluation over crafted benchmarks with varying levels of self-duality, we achieve average area reductions of 2.46%, 1.75%, and 1.76% compared to state-of-the-art logic optimization flows *compress2rs*, *dc2* and *dch*, respectively. Over cryptographic benchmarks, using the proposed XMG-based flow, area reductions up to 12% are achieved for benchmarks with high self-duality.
2) We propose state-of-the-art resubstitution and rewriting algorithms for XMGs. Our resubstitution algorithm uses a new filtering rule for 3-input Xor gates (Xor3). This filtering rule reduces the average runtime for resubstitution over the EPFL benchmarks by 52% while preserving the quality. Our rewriting algorithm for XMGs, called *exact XMG rewriting*, uses cut enumeration, NPN canonization, and exact synthesis. In contrast to the previous XMG rewriting approaches, the algorithm uses structural hashing to utilize existing logic and as such

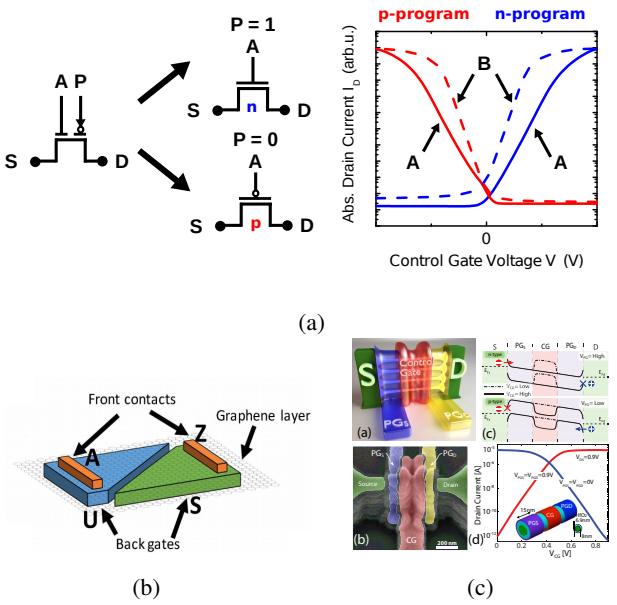


Fig. 1: Various emerging reconfigurable Nanotechnologies. (a) A generic RFET showing two gate terminals: The program (signal P) and the control gate (signal A) [30]. The Program gate controls the direction of the flow of charge carriers where as the control gate controls the flow of the charge carriers. The adjacent curve shows the V-shaped curve representing electrical symmetry for n- and p-type functionality. (b) It shows a graphene pn junctions where the back gates, (S and U) works as a control knob to control the ambipolarity. (c) It shows an all-around RFET called as *Three-Independent Gate FETs* (TIGFETs). The band diagrams are also shown here [25].

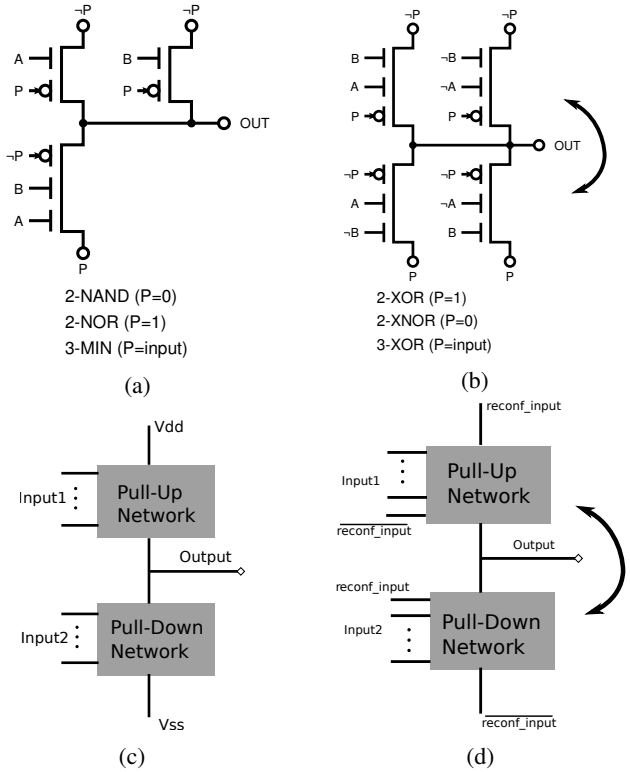


Fig. 2: Reconfigurable logic gate Minority and XOR3 demonstrated in [10, 30]. It shows how functionality changes with value of P. (c) Fixed Pull-up and pull-down network in case of Complimentary MOS logic gates. (d) Interchangeable Pull-up and pull-down network in case of RFET-based logic gates [33].

can achieve size reduction even if a smaller subnetwork is replaced with a larger one.

The remainder of the paper is organized as follows: In Section II, we introduce reconfigurable nanotechnologies, self-duality and previous works on XMGs. In Section III, we give the main motivation behind this work. This is followed by Section IV, which deals with details about the algorithm to generate benchmarks with varying degree of self-duality. In Section V, we discuss Boolean methods such as resubstitution as well as rewriting for XMGs. Section VI gives details about our experiments and analysis over various benchmark suites. Closing remarks are given in Section VII.

## II. Background

In this section, we introduce reconfigurable nanotechnologies and terminologies used throughout the paper.

### A. Reconfigurable nanotechnologies

Ambipolarity, as a phenomenon is observed at lower technology nodes, but often suppressed using process techniques [11]. The class of emerging nanotechnologies which aims to exploit this ambipolarity is often termed as emerging reconfigurable nanotechnology and the devices are called *reconfigurable field-effect transistors* (RFETs). These devices demonstrate both n- and p-type functionality from a single device on application of an external bias. Multiple device geometries based on various materials like silicon [10, 11], germanium [14], carbon [3] etc. have been proposed which exhibit near to full electrical symmetry in both n- and p-type functionality. This electrical symmetry is shown as V-shaped curve in Fig. 1a. Both 1-D devices (such as silicon or germanium nanowires [11] etc.) and 2-D devices (such as graphene p-n junctions [8], $WSe_2$ TIGFET [19] etc.) have been demonstrated to exhibit ambipolarity. Logic gates based on these devices are able to exhibit more than one functionalities as shown in Fig. 2.

### B. Self-dual functions

A logic function $f(x_1, \ldots, x_n)$ is said to be self-dual [12] if

$$f(x_1, \ldots, x_n) = \bar{f}(\bar{x}_1, \ldots, \bar{x}_n) \quad (1)$$

By complementing the function, an equivalent self-dual formulation is $\bar{f}(x_1, \ldots, x_n) = f(\bar{x}_1, \ldots, \bar{x}_n)$. For a particular instance of $x_1, \ldots, x_n$, $f(x_1, \ldots, x_n)$ and $\bar{f}(\bar{x}_1, \ldots, \bar{x}_n)$ are *dual* to each other.

**Theorem 1.** *There are* $2^{2^{n-1}}$ *different self-dual functions of n variables.*

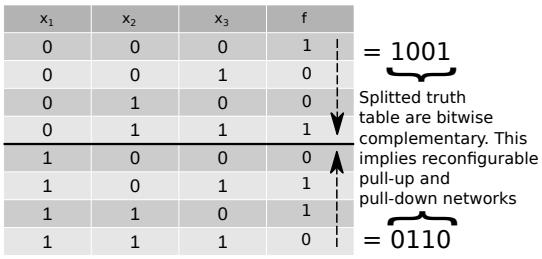


Fig. 3: Truth-Table for Xor3 logic gate. The truth-table is split over the value of $x_1$ which is the reconfigurable input.

*Proof.* For a self-dual function, since $f(x_1, \ldots, x_n) = \bar{f}(\bar{x}_1, \ldots, \bar{x}_n)$, only half of the inputs are sufficient to completely specify the function. From this, only $2^{n-1}$ combinations exist. Hence, total number of self-dual function for $n$ inputs considering both polarities (0 and 1) is equal to $2^{2^{n-1}}$. □

Fig. 3 shows a 3-input *Xor* function which is a self-dual function. Here, when the truth-table is divided over the value of $x_1$ (or any other arbitrary literal), the two half of the *Xor3* truth-table (*Xor2* and *XNor2* functions) are dual to each other.

In [33], the authors showed that self-dual functions are the logical abstraction for reconfigurable nanotechnology. The multiple functionality exhibited by RFET-based logic gates are due to the interchangeable pull-up and pull-down networks as shown in Fig. 2d. The switching of polarities of individual transistors in their respective pull-up and pull-down networks is caused by changing the potential at the program gate as shown in Fig. 2a and 2b. This change in potential at the program gate causes the PFET to become NFET and vice-versa causing the pull-up and pull-down networks to flip as shown in Fig. 2d. This corresponding switch in electrical behavior is abstracted conveniently in a self-dual function.

### C. Terminologies

We introduce some terminologies here, which will be used through the rest of the paper.

1) *Density of self-duality*: We define the term density of self-duality for a circuit or logic network as the ratio of total number of self-dual nodes to the total number of nodes.

2) *Trivial and non-trivial self-dual functions*: As shown in Theorem 1, self-dual functions are fewer (square root of total number of functions) as compared to non-self-dual functions. Moreover, among two-input functions, self-duality exists in those functions which are equivalent to either of the inputs or to their complements (for example, $f(a, b) = f(a)$). Such functions are implemented in circuits as wires and hence their implementation does not require any transistors. Thus two-input self-dual functions are termed here as *trivial functions* as they have no impact on the overall area of the circuit.

For functions with more than two inputs, their implementation with RFETs require fewer number of transistors as compared to their CMOS counterpart. These functions will have a direct impact on the area of the circuit. Hence, 3 or more input functions which are self-dual are termed as *non-trivial functions*.

### D. Earlier works on XMG

*Xor-Majority Graphs* in their current format, were first introduced in [22] as a means for underlying logic representation for exact synthesis. As exact synthesis uses SAT solving or enumeration, its runtime directly depends upon the size of the logic representation. Since XMGs have both binate (*Xor*) as well as unate (*Maj*) nodes, it gives a size-proportional logic representation for both $n$-input unate and binate functions as compared to the poor representation of binate logic (*Xor*-based logic) by MIGs or AIGs [28]. Algebraic optimizations for XMGs based logic synthesis was proposed in [28]. They explored Boolean algebraic optimizations for *Xor* and *Xor-Maj* logic and were able to achieve depth optimizations.

In the present work, we explore logic synthesis in order to maximize the self-duality within a circuit so that they can be efficiently mapped to RFETs. Since our objective is primarily post-mapping area, we are focusing on size optimization and hence, have not considered algebraic optimizations in the present work.

### E. NPN-Classification

Two functions $f(x)$ and $g(x)$ belong to the same NPN class if there exist a permutation $\sigma \in S_n$ and polarities $p_1, \ldots, p_n \in \mathbb{B}$ such that

$$f(x_1, \ldots, x_n) = g^p(x^{p_1}_{\sigma(1)}, \ldots, x^{p_n}_{\sigma(n)}). \tag{2}$$

This implies that functions $g$ can be made NPN equivalent to $f$ by either complementing the inputs, permuting the inputs or complementing the outputs or a combination of these. For example, functions $f_1 = x_1\bar{x}_2 + x_1x_3$ and $f_2 = x_1x_2 + \bar{x}_1x_3$ are NPN equivalent because the permutation of unate variables $x_2$ and $x_3$ lead to the same function. NPN classification is an effective method to represent Boolean functions as for a particular number of inputs, as NPN classes is a smaller subset of the overall Boolean functions possible.

## III. Motivation

Due to their device-level reconfigurability, RFETs allow efficient implementation of self-dual logic functions in terms of number of transistors [33]. For example, a Minority logic gate (shown in Fig. 2) needs 3 transistors in case of RFETs as compared to 10 transistors in CMOS technology [30]. This implies that circuit implementations with RFETs gain in area, if they have a higher density of non-trivial (3 or more input functions) self-dual gates. Hence, it is imperative that self-duality in a logic representation is preserved after logic optimizations.

From logic representation perspective, if we consider AIGs (the logic primitive is AND, which is a two-input primitive), a self-dual function will be broken into multiple AIG nodes and hence during logic optimization (which can use cut enumeration techniques), self-duality can be lost. Similarly, for MIG nodes, parity-based self-dual functions are not represented

**Algorithm 1:** Populating benchmarks with varying level of self-duality

**Data:** $num\_pis$, $levels$, $nodes\_per\_level$, $sdIndex$
**Result:** XMG network $N$

```
1  Set signalList ← [];
2  Set sd_or_normal ← 0;
3  for k ← 0 to num_pis do
4  |  signalList.add(N.create_pi());
5  for i ← 0 to levels do
6  |  for j ← 0 to nodes_per_levels do
7  |  |  fanins ← signalList.randSubSet();
8  |  |  if sd_or_normal < sdIndex then
9  |  |  |  node ← N.create_selfdual_gate(fanins);
10 |  |  else
11 |  |  |  node ← N.create_normal_gate(fanins);
12 |  |  signalList.add(node);
13 |  |  sd_or_normal ← (sd_or_normal + 1)
   |  |   mod 10 ;
14 for o ∈ signalList.not_used() do
15 |  N.create_po(o);
16 return N
```

TABLE I: Distribution of self-dual functions in NPN

| No of vars. | Functions | | NPN Classes | |
|---|---|---|---|---|
| | Self-dual + norm. | Total | Self-dual + norm. | Total |
| 1 | 1 + 3 | 4 | 1 + 1 | 2 |
| 2 | 4 + 12 | 16 | 1 + 3 | 4 |
| 3 | 16 + 240 | 256 | 3 + 11 | 14 |
| 4 | 256 + 64000 | 65536 | 7 + 215 | 222 |

in a compact manner which can again result in loss of self-duality during logic optimization [28].

In contrast, XMGs consists of both *Xor* and *Maj* nodes as logic primitives. Out of these, every majority and odd-inputs XOR functions are self-dual. Hence, using XMGs can better preserve self-duality during logic optimization as compared to other logic representatives.

## IV. Self-duality

### A. Self-duality in NPN classes

As stated in Theorem 1, self-dual functions are rare. TABLE I shows the distribution of the self-dual functions over all Boolean functions up to 4 variables and their NPN representatives. NPN canonization preserves the self-duality of a Boolean function, i.e., if a Boolean function is self-dual, so are all Boolean functions obtained by applying the NPN transformations to it. The numbers in the table illustrate that self-dual functions are not only rare when compared to the total number of Boolean functions, but also show that they reduce with increasing number of variables. Whereas 25% of the NPN representatives in 2 variables are self-dual, this percentage drops to 21.43% and 3.15% for 3 and 4 variables, respectively.

### B. Self-dual benchmarks

In order to evaluate the efficacy of our approach as compared to the state-of-the-art logic synthesis approaches, for RFET-based standard cell mapping, we generate benchmarks with varying density of self-dual logic gates within the circuit. We propose a metric called a *self-duality index* to vary the density of self-duality within a circuit. The rationale behind this is that if we have a larger number of self-dual components in the circuit, then more self-dual logic gates can be utilized during mapping. This leads to an improved area results for RFET-based circuit.

We generate multiple benchmarks using Algorithm 1. We start with an empty logic network and take four parameters as inputs– *number of Primary Inputs (PIs)* ($num\_pis$), *number of levels* ($levels$), *number of nodes per level* ($nodes\_per\_level$) and *self-duality index* ($sdIndex$, whose value has to be between 1 and 10). Depending upon the value of *self-duality index*, for every 10 nodes added in the logic network, number of self-dual nodes added, is equal to *self-duality index* (line 9) and the remaining (10− *self-duality index*) (line 11) nodes are normal nodes. By normal nodes, we mean adding logic nodes using AND, OR, XOR or constants while self-dual nodes implies adding Majority or 3-input XOR logic nodes. We maintain a signal list $SL$ where we keep adding all the newly created nodes (line 12). We then randomly select nodes from the signal list $SL$ to add to the circuit (line 7).

## V. Advanced Logic synthesis techniques

While the earlier works [22, 28] introduced the basic logic optimizations using XMGs, we extend the support of XMGs with advanced Boolean methods such as resubstitution as well as NPN-based cut-rewriting techniques.

### A. Boolean XMG resubstitution and filtering

Boolean resubstitution is a logic optimization method that re-expresses the function of a node $n$ in a logic network $N$ using nodes, called *divisors*, already present in $N$. Nodes that are exclusively used by $n$ and are not required by any other logic in the logic network become free and can be removed. A resubstitution leads to a size reduction if the number $k$ of newly added nodes to re-express a node's function is less than the number $l$ of removed nodes in its *maximum fanout-free cone* (MFFC, [5]).

Resubstitution algorithms are available for different multi-level logic representations including AIGs [4, 5], MIGs [26, 32], and logic networks [2, 9, 24] focusing on two-input And operations, three-input Majority operations, and combinations of two-input gates such as Xor-Ands, And-Xors, or Multiplexor-Xors, respectively.

Computing three-input Xor (Xor3 is a self-dual logic gate) resubstitutions is particularly time-consuming because divisor filtering techniques developed for And and Or operations cannot be applied. To substitute a node $n$ in a network with logic function $f_n(x)$ by a three-input Xor operation, three divisor nodes $d_1$, $d_2$, and $d_3$ have to be found, such that

$$f_n(x) = f_{d_1}(x) \oplus f_{d_2}(x) \oplus f_{d_3}(x) \tag{3}$$

**Algorithm 2:** Boolean filtering and resubstitution

```
Data: Window W in a logic network with root node n
Result: Node resubstitute for n or ⊥ if not resubstitution has
        been found
1  Set M ← W.computeMFFC(n);
2  Set D ← W.collectDivisors(n)\M;
3  Set TT ← W.simulate();
4  sortByDBP(D, TT, n);
5  for i ← 0 to |D| do
6      if 3 · DBP(D[i]) < DBP(n) then
7          return ⊥;
8      for j ← i + 1 to |D| do
9          if DBP(D[i]) + 2 · DBP(D[j]) < DBP(n) then
10             break;
11         for k ← j + 1 to |D| do
12             if f = TT[i] ⊕ TT[j] ⊕ TT[k] then
13                 return W.xor3_resub(n, D[i], D[j], D[k]);
14             if f = ¬TT[i] ⊕ TT[j] ⊕ TT[k] then
15                 return W.xor3_resub(n, D[i], D[j], D[k]);
16 return ⊥;
```

for all assignments to the primary inputs $x$, where $f_{d_1}$, $f_{d_2}$, $f_{d_3}$ are the divisor functions, respectively.

State-of-the-art Boolean resubstitution algorithms over-approximate the node functions using windowing to apply scalable truth-table computations. The algorithms have to iterate over all triples of nodes in a window of a root node $n$ (excluding the root node's MFFC) to test if Eq. 3 holds. The first substitution possible that reduces the network's size is accepted. In the worst case, if no resubstitution can be accepted, $\mathcal{O}(w^3)$ tests are required for a window with $w$ nodes.

Filtering techniques help to reduce the number of tests required and significantly speed-up the performance of resubstitution algorithms in practice. We develop a new filtering rule for three-input Xors guiding the search for divisors using distinguishing bit-pairs [6]: a resubstitution of a target node $n$ with function $f(x)$ and divisors nodes $d_1, d_2, d_3$ with functions $f_{d_1}(x), f_{d_2}(x), f_{d_3}(x)$ over common window inputs $x$ exists if and only if for any pair $\hat{x}_i \neq \hat{x}_j$ of input assignments

$$f(\hat{x}_i) \neq f(\hat{x}_j) \implies \bigvee_{1 \leq a,b \leq 3, a \neq b} d_a(\hat{x}_i) \neq d_b(\hat{x}_j). \quad (4)$$

Utilizing Eq. 4, we sort all divisor nodes in a window by the number of bit-pairs distinguished by the divisor with respect to the root node's target function. We define the *absolute distinguish bit power* $\mathrm{DBP}(n)$ of the root node $n$ as the number of pairs $(\hat{x}_i, \hat{x}_j)$ of input assignments for which $f_n(x_i) \neq f_n(x_j)$, and we define the *relative distinguishing bit power* $\mathrm{DBP}_n(d)$ of a divisor $d$ as the number of pairs $(\hat{x}_i, \hat{x}_j)$ of inputs assignments for which $f_n(\hat{x}_i) \neq f_n(\hat{x}_j)$ and $f_d(\hat{x}_i) \neq f_d(\hat{x}_j)$.

Algorithm 2 shows our Boolean filtering and resubstitution algorithm as pseudo code. The divisors are sorted (line 4) by their relative distinguishing bit power—higher relative distinguishing bit power will more likely lead to a possible resubstitution. We further leverage the relative distinguishing bit power to filter *insufficient* divisor triples. Given a sorted list $D = d_1, ..., d_w$ of divisors such that $\mathrm{DBP}_n(d_i) \geq \mathrm{DBP}_n(d_j)$ for all $i < j$, a single divisor $d$ can never be completed to divisor triple that passes the test in Eq. 3 if $3 \cdot \mathrm{DBP}_n(d) < \mathrm{DBP}(n)$ (line 6). Since the list is sorted, no remaining divisor will pass this test either such that the algorithm can terminate (line 7). For a similar reason, no divisor pair $d_i, d_j$, $i < j$, can be completed to a divisor triple that passes the test in Eq. 3 if $\mathrm{DBP}_n(d_i) + 2 \cdot \mathrm{DBP}_n(d_j) < \mathrm{DBP}(n)$ (line 9). In this case, the algorithm can proceed by selecting another candidate divisor $d_i$ (line 10).

### *B. Exact XMG rewriting*

Boolean rewriting is a logic optimization method that selects small parts of a logic network and replaces them with more compact implementations to reduce its number of nodes. State-of-the-art rewriting algorithms either rely on a database of precomputed size-optimum subnetworks for all Boolean functions up to 5 inputs [5] or compute size-optimum subnetworks on-the-fly using exact synthesis [31, 34]. DAG-aware rewriting [5], fast cut enumeration techniques [1], NPN canonization [13] of Boolean functions, and efficient caching [34] enable scalability.

Rewriting XMGs has been first proposed in [22] using a two-step approach: (1) A logic network is mapped into a network of $k$-feasible lookup-tables (LUTs); (2) the $k$-feasible LUTs are resynthesized into size-optimum XMGs. By repeating the two steps until convergence, substantial size reduction can be achieved.

We propose an improved XMG rewriting approach, called *exact XMG rewriting*, that integrates both steps into one algorithm. For each node, in the logic network, the set of all $k$-feasible cuts is enumerated, each cut is simulated to obtain its Boolean functions, and the functions are resynthesized using exact synthesis. In contrast to the previous approach, our algorithm takes advantage of structural hashing to utilize the existing logic within the network, such that a global size reduction can be achieved even if a locally smaller subnetwork is replaced with a larger subnetwork.

The algorithm can be parameterized with a set of gate primitives and supports synthesis of multiple candidates per cut function. A conflict limit in exact synthesis allows to limit the maximum synthesis effort per function. We consider exact XMG rewriting for three different sets of gate primitives:

1) Three-input Majority gates with two-input Xor gates as originally proposed by [22];
2) Three-input Majority gates and three-input Xor gates to enable a more compact representation of Boolean functions. Note that with constants the three-input Xor gate can simulate the function of two-input Xor gates and, thus, is a generalization of two-input Xor; and
3) Three-input Majority gates without constants and three-input Xor gates to improve the internal self-duality of a logic network during rewriting.

TABLE II: Runtime improvement in resubstitution using our filtering rule

| Benchmark | Size Before | runtime with filter | runtime without filter | Improvement |
|---|---|---|---|---|
| adder | 1020 | 0.21 | 0.31 | 32.26 |
| bar | 3336 | 0.9 | 2.34 | 61.54 |
| div | 57247 | 35.58 | 58.85 | 39.54 |
| hyp | 214335 | 321.19 | 391.29 | 17.92 |
| log2 | 32060 | 16.26 | 28.38 | 42.71 |
| max | 2865 | 0.89 | 1.39 | 35.97 |
| multiplier | 27062 | 13.44 | 33.53 | 59.92 |
| sin | 5416 | 3.34 | 7.49 | 55.41 |
| sqrt | 24618 | 13.64 | 29.77 | 54.18 |
| square | 18484 | 6.78 | 17.77 | 61.85 |
| arbiter | 11839 | 4.11 | 6.08 | 32.40 |
| cavlc | 693 | 7.28 | 55.09 | 86.79 |
| ctrl | 174 | 0.64 | 5.97 | 89.28 |
| dec | 304 | 0.02 | 0.02 | 0.00 |
| i2c | 1342 | 0.34 | 1.38 | 75.36 |
| int2float | 260 | 0.19 | 1.32 | 85.61 |
| mem_ctrl | 46836 | 23.24 | 47.18 | 50.74 |
| priority | 978 | 0.34 | 1.15 | 70.43 |
| router | 257 | 0.09 | 0.2 | 55.00 |
| voter | 13758 | 5.46 | 10.81 | 49.49 |
| **Average** | | | | **52.82** |

## VI. Experiments and Discussion

In the present section, we evaluate our approach with various experiments. All the above implementations are carried out in *mockturtle* from EPFL logic synthesis libraries [27].

### A. Methodology

We extended the XMG network and implemented Boolean methods such as resubstitution and exact XMG rewriting techniques within *mockturtle*. We first evaluate the speedup in runtime due to the filtering rule implemented with XMG resubstitution algorithm. We then apply our XMG-based flow over the generated benchmarks, the cryptography benchmarks and the EPFL benchmarks. To produce area results, we use the standard mapper with ABC logic synthesis framework with the logic gates as proposed in [30]. Further, we compare our flow with the state-of-the-art scripts within ABC such as *compress2rs*, *dc2* and *dch*.

### B. Runtime improvement with filtering rule in resubstitution

In order to measure the improvement in runtime using the Xor3-based filtering rule, we carry out one iteration of resubstitution (with and without filtering) over EPFL benchmarks using XMGs. The third and the forth column in TABLE II shows the runtime for our resubsitution algorithm for both the flows. We can see that on average we get $50\%$ improvement in the runtime across all benchmarks.

### C. Analysis on crafted self-dual benchmarks

Within this experiment we use Algorithm 1 to populate multiple benchmarks with varying numbers of PIs, numbers of levels and numbers of nodes per levels. The self-duality index is taken care by the variable *sdIndex* whose value is iterated from 1 to 10 to populate 10 benchmarks for a single set of parameters. We apply resubstitution as well as NPN-based cut-writing over XMG till convergence and then carry out standard-cell mapping using RFET-centric generic library and compare the results across various ABC scripts. In all these experiments, we converge after one iteration. Hence, it is fair enough to compare with scripts from ABC.

TABLE III shows the comparison of post-mapping area carried out using different scripts for our crafted benchmarks. The first column shows the value of sd-index which signifies how many self-dual nodes have been added for every 10 nodes. The columns *xmg-c2rs*, *xmg-dc2*, and *xmg-dch* show how much the final area using XMG-based optimization compares with the ABC logic optimization flows. The numbers are in percentage where a positive value means that XMG gives better numbers as compared to the ABC scripts and vice-versa. The final two columns (sd_ratio and sd_ratio') are the density of self-duality for the initial XMG after Algorithm 1 and final XMG after optimization using Boolean methods respectively. For XMG-based flow one can notice a direct correlation between the improvements in area and the higher values for self-duality ratios. Further, *dch* script shows a large variation across sd-index from 1 to 10, as it gives much better numbers for smaller values of sd-index, but gives worse numbers as the sd-ratio gets more than 50%. *Compress2rs* script gives the closest result across the sd-index. Hence, this experiment shows that with increase in the sd-index, XMG based optimization gives better numbers as compared to the state-of-the-art ABC scripts.

TABLE III: Comparison of final area with respect to ABC scripts of *compress2rs*, *dc2* and *dch*

| Sd-index | xmg-c2rs (%) | xmg-dc2 (%) | xmg-dch (%) | sd-ratio | sd-ratio' |
|---|---|---|---|---|---|
| 1 | 0.10 | - 1.72 | -3.74 | 28.62 | 36.14 |
| 2 | 1.83 | 0.16 | -1.13 | 32.47 | 40.23 |
| 3 | 2.50 | 0.78 | 0.15 | 36.89 | 44.24 |
| 4 | 3.09 | 1.59 | 1.05 | 41.76 | 48.82 |
| 5 | 3.20 | 2.01 | 1.73 | 47.05 | 53.41 |
| 6 | 3.22 | 2.34 | 2.32 | 53.09 | 58.52 |
| 7 | 3.24 | 2.68 | 3.17 | 59.95 | 64.53 |
| 8 | 3.00 | 2.94 | 3.67 | 68.00 | 71.37 |
| 9 | 2.78 | 3.18 | 4.48 | 76.87 | 79.35 |
| 10 | 1.66 | 3.63 | 5.95 | 100.00 | 100.00 |
| **Average** | 2.46 | 1.75 | 1.76 | | |

### D. Analysis on cryptography protocol benchmarks

While the previous experiment was conducted on crafted benchmarks, it is imperative to evaluate our approach on actual benchmark suites. For this, we conducted experiments comparing our XMG-based approach with various optimization scripts from ABC in order to establish our conjecture that an increase in self-duality within a circuit can be better optimized with XMGs. These benchmarks were taken from high-level cryptography protocols such as *Fully Homomorphic Encryption*

TABLE IV: Comparison of mapped area for the cryptography benchmarks using XMG optimization as compared to ABC scripts

| Benchmark | sd-ratio | init_area | c2rs_area | dc2_area | dch_area | Best Area ABC | XMG Area | xmg-best(%) |
|---|---|---|---|---|---|---|---|---|
| AES-expanded | 22.23 | 98145.5 | 87771.5 | 87791.5 | **87363.5** | 87363.5 | 87687.5 | -0.37 |
| AES-non-expanded | 21.92 | 119367.5 | **105743.5** | 111298 | 110207.5 | 105743.5 | 110121 | -4.14 |
| DES-expanded | 43.39 | 48138 | **40409.5** | 41843.5 | 41859 | 40409.5 | 46820.5 | -15.87 |
| DES-non-expanded | 40.67 | 48382.5 | **40609.5** | 41712.5 | 41464.5 | 40609.5 | 45646.5 | -12.40 |
| adder_32bit | **85.29** | 285 | 285 | 285 | 285 | 285 | 285 | 0.00 |
| adder_64bit | **81.42** | 573 | 573 | 573 | 573 | 573 | 573 | 0.00 |
| adder | **99.21** | 1690.5 | 1149 | 1149 | 1149 | 1149 | 1149 | 0.00 |
| comparator_32bit_signed | 40.65 | 325 | 255 | 250 | 245 | **245** | 281 | -14.69 |
| md5 | **55.87** | 112702.5 | 114325 | 114805 | 114505 | 114325 | **99624** | 12.86 |
| mult_32x32 | 41.77 | 11293 | 11011 | **9925.5** | 10344.5 | 9925.5 | 12341.5 | -24.34 |
| Sha-1 | **63.47** | 161993.5 | 161578.5 | 163646.5 | 164229 | 161578.5 | **141114.5** | 12.67 |
| Sha-256 | **71.76** | 287858.50 | 281668.50 | 285691.00 | 276010.00 | 276010.00 | **248885.50** | 9.83 |

TABLE V: Comparison of mapped area for the EPFL benchmarks using XMG optimization as compared to ABC scripts

| Benchmarks | initial_area | c2rs_area | dc2_area | dch_area | XMG_area | sd-ratio | sd-ratio' |
|---|---|---|---|---|---|---|---|
| adder | 1149 | 1149 | **1149** | 1149 | 1149 | 98.837 | **99.21875** |
| bar | 4633 | 4147 | 5565.5 | 3858 | 6681.5 | 48.324 | 49.38 |
| div | 110964 | 50550.5 | 48817.5 | 37992.5 | 88045 | 28.74 | 36.5665 |
| hyp | 327233 | 316630.5 | 305032 | 0 | 369719 | 56.1501 | 56.8566 |
| log2 | 47188.5 | 45714.5 | 46608.5 | 45861 | 50822.5 | 43.9896 | 44.444 |
| max | 4408.5 | 4558 | 4280 | 4358.5 | 5354.5 | 50.575 | 50.571 |
| multiplier | 37268 | 37384.5 | 37207.5 | 37831 | 39937 | 50.107 | 50.04 |
| sin | 9131.5 | 9450.5 | 9169.5 | 9072 | 9191.5 | 43.07 | 44.169 |
| sqrt | 54239.5 | 28765.5 | 32655.5 | 42901.5 | 56764 | 38.869 | 39.0425 |
| square | 27180 | 24265 | 24708 | 23993.5 | 29478.5 | 54.282 | 56.63 |
| arbiter | 18014.5 | 18014.5 | 18048 | 18049.5 | 26644.5 | 63.3462 | 63.3486 |
| cavlc | 1118.5 | 1074 | 1071 | 1042.5 | 1504 | 47.471 | 51.615 |
| ctrl | 236.5 | 182.5 | 175 | 172 | 237 | 43.801 | 50.535 |
| dec | 464 | 464 | 464 | 464 | 554 | 9.356 | 10.0344 |
| i2c | 2116 | 1847 | 1936 | 1775.5 | 2796 | 45.608 | 46.5456 |
| int2float | 407 | 348 | 346.5 | 341.5 | 539 | 46.987 | 48.936 |
| mem_ctrl | 76604 | 72688 | 70491.5 | 67965 | 92565.5 | 52.473 | 53.2045 |
| priority | 1208 | 973 | 880.5 | 1311.5 | 1287 | 44.744 | 45.55 |
| router | 449 | 364.5 | 466.5 | 457.5 | 566 | 38.942 | 39.6004 |
| voter | 19664.5 | 12209.5 | 14605 | 14961 | **11726.5** | 54.4 | **71.437** |

(FHE) and secure *Multy-Party Communication* (MPC) [15, 21][1]. These benchmark suite contains circuits ranging from block ciphers (AES and DES) and hash functions such as (MDA-5 and SHA) to arithmetic functions (adders and comparators).

The results are shown in TABLE IV. As in the case of crafted benchmarks, here also we compare the post-mapping area. We compare our XMG-based approach to various ABC based optimization scripts. The first column shows the sd-ratio which implies the density of self-duality in the circuit. One can notice that most of the benchmarks from the cryptography domain have high density of self-dual gates ($>50\%$), particularly parity functions as it is an integral logic function in any cryptographic applications. For benchmarks, *md5*, *SHA-1* and *SHA-256*, our XMG-based approach outshines other ABC-based scripts. This is also coherent with their sd-ratio values which are high. Our XMG-based approach gives up to $12\%$ smaller area as compared to the other flows. For smaller benchmarks, such as adder, all the flows reach the optimal area numbers.

[1]The benchmarks were obtained from https://homes.esat.kuleuven.be/nsmart/MPC/

For *AES-non-expanded*, *compress2rs* gives the best result, which is 4% better than our XMG-based approach. In this case as well our XMG-based approach fares well in comparison to *dc2* and *dch* scripts. In case of benchmarks, where sd-ratio is lesser, the XMG-based approach gives poor results. This is due to the fact, that XMG representation are bigger as compared to other 2-input primitives. An important consideration here is that since we use ABC as the standard-cell mapper, it converts all three-input Majority primitives and Xor primitives to multiple two-input And primitives. During technology mapping, it carries out cut-enumeration which can suppress self-duality within the circuit. Hence, the technology mapping of the XMG-based networks can get stuck in local minimum and can lead to poor mapping results. A mapper with self-duality as target metric can lead to better numbers but is beyond the scope of this work.

### *E. Analysis on EPFL Benchmarks*

Finally, we evaluate our approach over the EPFL benchmarks. The results are shown in TABLE V. However, unlike

cryptography protocol benchmarks, the area results for XMG-based approach are worse as compared to ABC-based scripts due to low self-duality density as shown by the sd-ratio values. An exception here is the *arbiter* benchmark. While technology mapping of *arbiter* using ABC-based scripts reports three gates–Nand, Nor and Inv, technology mapping on XMG-based flow reports use of an additional Minority gate. This clearly implies that the technology mapper is stuck in some local minimum and reports worse area numbers for XMG-based flow due to reasons stated before. However, for *voter* benchmark, which shows a higher percentage of self-duality, the post-mapping area using XMG-based approach is the best of all other flows which is coherent to our conjecture. Hence, our approach holds true for benchmarks with higher self-duality.

## VII. Conclusion

The present work explores logic synthesis from an emerging nanotechnology perspective. Keeping a particular goal in mind that the self-dual logic functions are implemented efficiently with RFETs, we explore XMGs as logic representation so as to exploit self-duality in circuit. We have considered XMGs due to two reasons: (i) they provide a compact logic representation and (ii) both majority and odd input parity function are self-dual which can be efficiently represented by XMGs. We develop advance Boolean methods such as resubstitution and rewriting techniques for XMGs to get powerful optimizations. We show that circuits which have high density of self-duality show better area results for XMG-based approach as compared to the state-of-the-art ABC optimization flows. We further evaluate the XMG-based approach using circuits from cryptography and EPFL benchmarks. An XMG based technology mapper with a target metric for self-duality is a promising area as a future research direction for the commercial development of emerging reconfigurable nanotechnologies.

## References


[1] Jason Cong, Chang Wu, and Yuzheng Ding. "Cut Ranking and Pruning: Enabling a General and Efficient FPGA Mapping Solution". In: *ISFPGA*. 1999. DOI: 10.1145/296399.296425.

[2] Victor N. Kravets and Prabhakar Kudva. "Implicit enumeration of structural changes in circuit optimization". In: *DAC*. 2004. DOI: 10.1145/996566.996691.

[3] Yu-Ming Lin et al. "High-performance Carbon Nanotube Field-effect Transistor with Tunable Polarities". In: *IEEE Trans. Nanotechnol.* (2005). DOI: 10.1109/TNANO.2005.851427.

[4] Alan Mishchenko and Robert K. Brayton. "Scalable logic synthesis using a simple circuit structure". In: *IWLS*. 2006.

[5] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. "DAG-aware AIG rewriting a fresh look at combinational logic synthesis". In: *DAC*. 2006. DOI: 10.1145/1146909.1147048.

[6] Kai-Hui Chang, Igor L. Markov, and Valeria Bertacco. "Fixing Design Errors With Counterexamples and Resynthesis". In: *IEEE TCAD* (2008). DOI: 10.1109/TCAD.2007.907257.

[7] Robert Brayton and Alan Mishchenko. "ABC: An Academic Industrial-Strength Verification Tool". In: Berlin, Heidelberg, 2010. ISBN: 978-3-642-14295-6. DOI: 10.1007/978-3-642-14295-6_5.

[8] S. Tanachutiwat et al. "Reconfigurable multi-function logic based on graphene p-n junctions". In: *DAC*. 2010. DOI: 10.1145/1837274.1837496.

[9] Alan Mishchenko et al. "Scalable don't-care-based logic optimization and resynthesis". In: *ACM TRECTS.* (2011). DOI: 10.1145/2068716.2068720.

[10] André Heinzig et al. "Reconfigurable silicon nanowire transistors". In: *Nano Letters* (2012). DOI: 10.1021/nl203094h.

[11] M. De Marchi et al. "Polarity control in double-gate, gate-all-around vertically stacked silicon nanowire FETs". In: *IEDM*. 2012. DOI: 10.1109/IEDM.2012.6479004.

[12] Tsutomu Sasao. *Switching theory for logic synthesis*. Springer Science & Business Media, 2012.

[13] Zheng Huang et al. "Fast Boolean matching based on NPN classification". In: *FPT*. 2013. DOI: 10.1109/FPT.2013.6718374.

[14] Jens Trommer et al. "Material Prospects of Reconfigurable Transistor (RFETs)–From Silicon to Germanium Nanowires". In: *MRS* (2014).

[15] Martin R Albrecht et al. "Ciphers for MPC and FHE". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015.

[16] L. Amarú et al. "New Logic Synthesis as Nanotechnology Enabler". In: *Proceedings of the IEEE* (2015). DOI: 10.1109/JPROC.2015.2460377.

[17] Jens Trommer et al. "Functionality-Enhanced Logic Gate Design Enabled by Symmetrical Reconfigurable Silicon Nanowire Transistors". In: *IEEE Trans. Nanotech.* (2015). DOI: 10.1109/TNANO.2015.2429893.

[18] L. Amarú, P. E. Gaillardon, and G. De Micheli. "Majority-Inverter Graph: A New Paradigm for Logic Optimization". In: *TCAD* (2016). ISSN: 0278-0070. DOI: 10.1109/TCAD.2015.2488484.

[19] Giovanni V. Resta et al. "Polarity control in WSe2 double-gate transistors". In: *Scientific Reports* (2016).

[20] M. T. Bohr and I. A. Young. "CMOS Scaling Trends and Beyond". In: *IEEE Micro* (2017). DOI: 10.1109/MM.2017.4241347.

[21] Melissa Chase et al. "Post-quantum zero-knowledge and signatures from symmetric-key primitives". In: *ASCCCS*. 2017.

[22] W. Haaswijk et al. "A novel basis for logic rewriting". In: *ASP-DAC*. 2017. DOI: 10.1109/ASPDAC.2017.7858312.

[23] I. Háleček, P. Fišer, and J. Schmidt. "Are XORs in logic synthesis really necessary?" In: *DDECS*. 2017. DOI: 10.1109/DDECS.2017.7934583.

[24] Luca Gaetano Amarù et al. "Improvements to Boolean resynthesis". In: *DATE*. 2018. DOI: 10.23919/DATE.2018.8342108.

[25] Shubham Rai et al. "Emerging Reconfigurable Nanotechnologies: Can They Support Future Electronics?" In: *ICCAD*. 2018. DOI: 10.1145/3240765.3243472.

[26] Heinz Riener et al. "Size Optimization of MIGs with an Application to QCA and STMG Technologies". In: *NANOARCH*. 2018. DOI: 10.1145/3232195.3232202.

[27] Mathias Soeken et al. *The EPFL Logic Synthesis Libraries*. 2018. arXiv: 1805.05121 [cs.LO].

[28] Zhufei Chu et al. "Structural Rewriting in XOR-Majority Graphs". In: *ASPDAC*. New York, NY, USA, 2019. DOI: 10.1145/3287624.3287671.

[29] Gage Hills et al. "Modern microprocessor built from complementary carbon nanotube transistors". In: *Nature* (2019).

[30] S. Rai et al. "Designing Efficient Circuits Based on Runtime-Reconfigurable Field-Effect Transistors". In: *TVLSI* (2019). ISSN: 1063-8210. DOI: 10.1109/TVLSI.2018.2884646.

[31] Heinz Riener et al. "On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis". In: *DATE*. 2019. DOI: 10.23919/DATE.2019.8715185.

[32] Heinz Riener et al. "Scalable Generic Logic Synthesis: One Approach to Rule Them All". In: *DAC*. 2019. DOI: 10.1145/3316781.3317905.

[33] S. Rai, M. Raitza, and A. Kumar S.S. Sahoo. "DiSCERN: Distilling Standard-Cells for Emerging Reconfigurable Nanotechnologies". In: *DATE*. 2020.

[34] Heinz Riener, Alan Mishchenko, and Mathias Soeken. "Exact DAG-aware Rewriting". In: *DATE*. 2020.