Utilizing XMG-based Synthesis to Preserve Self-Duality for RFET-Based Circuits

Shubham Rai*, Alessandro Tempia Calvino†, Heinz Riener†, Giovanni De Micheli†, Akash Kumar*
†Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
*Chair for Processor Design, CfAED, Technische Universität Dresden, Germany

***Abstract*—Individual transistors based on emerging reconfigurable nanotechnologies exhibit electrical conduction for both types of charge carriers. These transistors (referred to as *Reconfigurable Field-Effect Transistors* (RFETs)) enable dynamic reconfiguration to demonstrate either a p- or an n-type functionality. This duality of functionality at the transistor level is efficiently abstracted as a self-dual Boolean logic, that can be physically realized with fewer RFET transistors compared to the contemporary CMOS technology. Consequently, to achieve better area reduction for RFET-based circuits, the self-duality of a given circuit should be preserved during logic optimization and technology mapping. In this paper, we specifically aim to preserve self-duality by using *Xor-Majority Graphs* (XMGs) as the logic representation during logic synthesis and technology mapping. We propose a synthesis flow that uses new restructuring techniques, called *rewriting* and *resubstitution* for XMGs to preserve self-duality during technology-independent logic synthesis. For technology mapping, we use a novel open-source and a logic-representation agnostic mapping tool. Using the above-proposed XMG-based flow, we demonstrate its benefits by comparing post-mapping area for synthetic and cryptographic benchmarks with three different synthesis flows: (i) AIG-based optimization and AIG-based mapping; (ii) XMG-based optimization with AIG-based mapping; (iii) AIG-based optimization with logic-representation agnostic mapping. Our experiments show that the proposed XMG-based flow efficiently preserves self-duality and achieves the best area results for RFET-based circuits (up to 12.36% area reduction) with respect to the baseline.**

I. Introduction

Emerging reconfigurable nanotechnologies offer a unique feature set over the traditional CMOS technologies. The transistors based on reconfigurable nanotechnologies can be dynamically reconfigured to exhibit either a p- or an n-type polarity [1], [2], [3]. Hence, these transistors are termed as *Reconfigurable Field-Effect Transistors* (RFETs) [4]. Such reconfigurable properties can be utilized in circuits to achieve more functionality per computational unit [4], [5]. Recently, it has been shown that RFETs allow efficient implementation of *self-dual* logic functions with few transistors [6]. Therefore, to achieve better area results for RFETs-based circuits, synthesis flow should utilize the maximum available self-duality in the circuit. The present work focuses on improving the logic synthesis and the technology mapping flow to achieve better area results for circuits based on emerging reconfigurable nanotechnologies by preserving the self-duality of a given circuit.

Logic synthesis plays an important role in optimizing a logic representation for a given circuit in terms of a cost function, typically focusing on the reduction of area or delay. At the technology-independent level, multi-level logic representations are used to represent and optimize circuits. Recently, novel representations have been proposed that enhance *And-Inverter Graphs* (AIGs) [7] and *Majority-Inverter Graphs* (MIGs) [8] with an additional XOR primitive. These new logic representations, called *Xor-And Graphs* (XAGs) [9] and *Xor-Majority Graphs* (XMGs) [10], [11], offer better compaction and, thus, often have a positive effect on the performance of logic minimization techniques [10]. Technology mapping on the other hand, focuses on expressing the minimized logic representation in terms of a network of logic gates chosen from a given library [12].

We explore an XMG-based synthesis flow for reconfigurable nanotechnologies to preserve and utilize the existing self-duality of a circuit, as they are less prone to disrupting self-duality compared to AIGs. XMG's gate primitives – XOR and MAJORITY offer a compact representation to abstract self-dual logics, because MAJORITY and odd-input XORs are self-dual.

We provide experimental evidence that a logic representation that compactly abstracts self-duality is able to preserve more self-duality in the circuit and achieve better area results for RFETs-based circuits. This is motivated by the fact that synthesis approaches such as logic optimization and technology mapping involve structural changes over a graph representation of a circuit which can disrupt the existing self-duality of a circuit. Our experiments show that for circuits with high percentage of self-dual logic (for ex. Majority and odd-input XORs), AIGs are more prone to decompose self-dual logic and preserve less self-dual logic as compared to XMGs.

The present work is an extension of the prior work [13]. While the prior work [13] did explore XMG-based logic synthesis for RFETs-based circuits, it suffered from the lack of a logic representation-agnostic technology mapper. Hence, in this work, we extend and use the recently proposed versatile mapper [14] that can perform technology mapping natively on XMGs to solve the limitation of the prior work [13]. The technology-independent mapper can use XMGs as the subject graph, and therefore our approach avoids the suboptimal process of converting to other logic representations that break down self-dual logic gates into smaller primitives. Further, we carry a detailed evaluation in terms of self-dual cuts recognized by the AIG and the XMG flows. Additionally, we propose a synthesis flow meant specifically for RFETs-based circuits and

evaluate it with three other flows. The scope of the present work is limited to logic synthesis and technology-independent mapping and does not focus on the physical synthesis of RFETs-based circuits. The major contributions are as follows:

- To preserve self-duality, we propose an XMG-based synthesis flow for circuits based on reconfigurable nanotechnologies. The XMG-based synthesis flow enables a better area reduction for RFET-based circuits.
- We propose a resubstitution and a rewriting algorithm for XMGs. We demonstrate that the two techniques play an important role for XMG optimization and also increase the self-duality density of the circuits.
- Our resubstitution algorithm uses a new filtering rule for 3-input XOR gates (XOR3) which drastically reduces the runtime.
- Using an area-oriented mapping, we demonstrate that as compared to AIGs, XMGs enable a higher percentage of self-dual cuts during mapping. The higher share of self-dual cuts subsequently leads to more mapping opportunities on to self-dual logic gates.

We compare our XMG-based approach with three different flows. First, we compare it to the native flow of the state-of-the-art tool ABC [15] (baseline). Second, we use XMG-based logic synthesis as a starting point and compare the area results calculated by the logic-representation agnostic mapper [14] with those calculated by the ABC technology mapper. Third, since the logic-representation agnostic mapper can support different logic representations, we compare our XMG-based approach with the AIG-based approach within the *mockturtle* framework [16].

We perform experiments over two sets of benchmarks to evaluate our approach. In the first set of experiments, we enumerate synthetic benchmarks with varying degrees of self-duality and show that our XMG-based approach gives the best results across these three different flows. In the second set of experiments, we use cryptographic benchmarks [17], [18]. We first demonstrate the impact of logic optimization techniques on the runtime and self-duality of the graph representation of circuits. We show that the XOR3 filtering rule leads to 59.48% improvement in runtime. In conjunction, the two techniques– resubstitution and rewriting, lead to a 23% average increment of self-duality or the cryptographic benchmark suite. We then show that our proposed XMG-based approach leads to an area savings of up to 12.3% with respect to the baseline. Last, we explore the relation between self-dual cuts and the final share of area by self-dual logic gates as a percentage of the overall area. We can see that on an average, the XMG-based approach results in 7.66% more area occupied by self-dual gates compared to the AIG-based approach within *mockturtle* framework. These comparisons show that for circuits with higher self-duality, our XMG-based flow achieves superior results as compared to other contemporary flows.

The remainder of the paper is organized as follows: Section II presents a motivating example of a simple circuit and compares the AIG and the XMG flow. Section III introduces reconfigurable nanotechnologies and explains how self-duality is a natural abstraction for logic gates based on these emerging technologies. In Section IV, we explain how self-duality can be preserved using XMG-based logic synthesis and technology mapping. This is followed by Section V, which explains the proposed resubstitution and the rewriting algorithm for XMGs. Then, in Section VI, we brief about our versatile mapper. This is followed by Section VII, which presents details about the algorithm to generate benchmarks with varying degrees of self-duality. Section VIII contains details about our experimental analysis. Closing remarks can be found in Section IX.

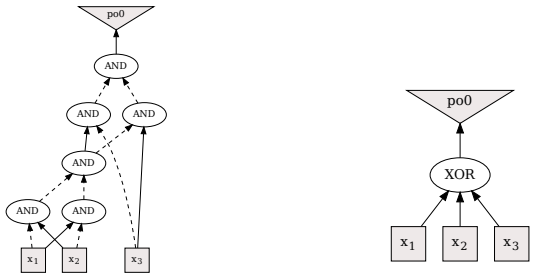
II. Motivation

In order to understand the motivation behind this work, let us consider a circuit that consists of a single 3-input XOR. Our objective is to carry out technology mapping of this given circuit. Two straightforward mappings for the given circuit are possible– one where the circuit is mapped to two 2-inputs XOR gates; and the other directly to a 3-input XOR gate. From a CMOS perspective, the mapper should prefer the first mapping as we know that that a 3-input XOR logic is avoided in CMOS as it requires multiple transistors and often has large delay [19]. Logic gates with many CMOS transistors require cascading or branching of multiple transistors that hampers the performance of such logic gates based on CMOS. Hence, circuits based on CMOS prefer logic gates with few inputs as such logic gates are better in terms of performance and area. This is also one of the considerations in contemporary logic representation of AIG as CMOS favours negative unate logic [20]. However, in the case of RFETs, a 3-input XOR is a preferred mapping since it is a self-dual logic gate [6]. From a logic synthesis perspective, the contemporary logic representation of AIG uses 6 nodes and 13 edges to represent the circuit. In contrast, XMG uses 1 nodes and 4 edges. Both the AIG and XMG representations are shown in Figure 1. If we consider the AIG representation, a simple 3-input cut-based technique during logic optimization and technology mapping results in the first kind of mapping. However, in the case of XMGs, with the same setting, the second mapping is achieved. The higher number of edges and nodes in AIG compared to XMG explains this difference in mapping results. This simple intuitive example motivates us to explore XMGs for RFETs-based circuit. As CMOS favors negative unate logic gates, AIGs are the natural abstraction for CMOS logic [20]. However, are AIGs appropriate for RFETs-based circuits as well? In this paper, we investigate this question.

III. Background

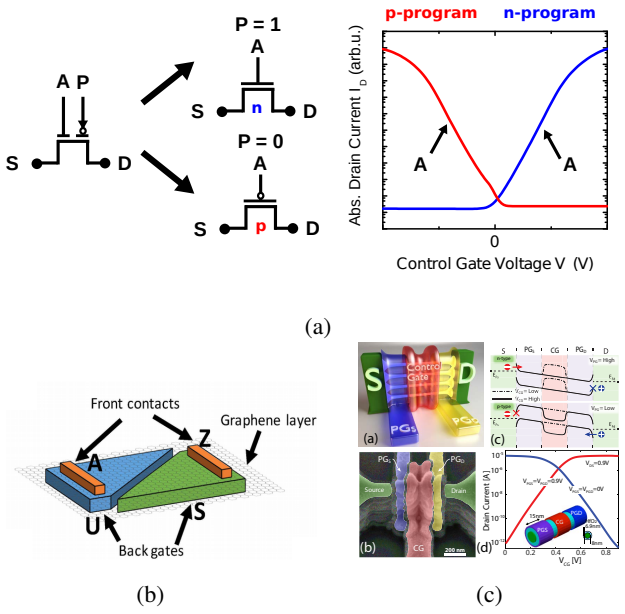
A. Reconfigurable nanotechnologies

Ambipolarity, or ambipolar conduction, is a phenomenon observed at lower technology nodes where the transistor allows conduction of both the charge carriers through the channel. Ambipolar conduction is enhanced using process techniques for several nanoscale FET devices based on materials such as silicon or germanium [22]. The class of emerging nanotechnologies that aims to take advantage of this ambipolarity is termed reconfigurable nanotechnology and the devices are



(a) AIG (b) XMG

Fig. 1: Edges and nodes representation in case of AIG and XMG representation for the function, $f = x_1 \oplus x_2 \oplus x_3$ (3-input XOR). The AIG has 6 gates and 13 edges, while the XMG has 1 gate and 4 edges.



(a) (b) (c)

Fig. 2: A generic RFET showing two gate terminals: The program (signal P) and the control gate (signal A) [19]. The program gate controls the type of charge carriers whereas 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 p-n junction where the back gates (S and U) work as a control knob to control the ambipolarity. (c) It shows an all-around RFET, called *Three-Independent Gate FETs* (TIGFETs). The band diagrams are shown in [21].

referred to as *reconfigurable field-effect transistors* (RFETs). These devices demonstrate both n- and p-type functionality from a single device when an external bias is applied. Multiple device geometries based on various materials like silicon [23], [22], germanium [24], carbon [25] 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. 2. Both 1-D (such as silicon [23] or germanium nanowires [22] etc.) and 2-D devices (such as graphene p-n junctions [26], WSe_2 TIGFET [27] etc.) have been demonstrated to exhibit ambipolarity. In terms of feasibility, silicon or germanium nanowires-based RFETs follow a similar manufacturing and fabrication process as CMOS [28], [29] and hence, they are closer to commercial integration.

Further, RFETs allow multiple gate terminals on a single channel, thereby reducing the on-channel resistance. This enables a *wired-AND* functionality [30] for multiple input gate terminals that can be used to realize large logic gates with more than 2 inputs. This has been shown experimentally in [31], [32]. Having multiple gate terminals on a single channel enables designing large logic gates with more than 2 inputs without compromising on performance [32]. More details about the electrical performance and the physics of RFETs can be found in [4], [28], [19].

B. RFETs-based logic gates

Ambipolarity at the transistor level can be exploited for designing efficient logic gates based on RFETs [19]. Since individual transistors allow dynamic reconfiguration between p and n-type behavior, logic gates can exploit such dynamic reconfigurability to have more than one functionality [33], [19], [34] as shown in Fig. 3. It can be seen from the figure, that a single circuit schematic can realize up to three different logic functions.

Unlike CMOS, different functionalities such as NAND and NOR functionality demonstrate equal current drive strength [4]. It is to be noted that logic gates exhibiting multiple functionalities follow a similar schematic of pass-transistor logic [32], [4]. Exhibiting dynamic reconfiguration between logic functionalities also has a delay overhead as demonstrated in [19] as compared to the static logic gate design. However, equal current drives in both p- and n-type configuration, and having multiple gate terminals on a single channel enable better performance and better compaction in terms of area as compared to the CMOS logic gates with more than 2 inputs [35], [32].

C. Self-dual functions

The *dual* of an n-input Boolean function $f(x_1, \ldots, x_n)$ over the Boolean variables $x_1, \ldots, x_n$ is given by $\overline{f(\bar{x}_1, \ldots, \bar{x}_n)}$, i.e., the dual is obtained first by replacing each literal x_i with $\bar{x}_i$ and then by complementing the function. We write $f^d(x_1, \ldots, x_n)$ to denote the dual of a Boolean function $f(x_1, \ldots, x_n)$.

A logic function $f(x_1, \ldots, x_n)$ is called *self-dual* [36] if

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

or, equivalently, if

$$\overline{f(x_1, \ldots, x_n)} = f(\bar{x}_1, \ldots, \bar{x}_n) \tag{2}$$

for all $x_1, \ldots, x_n \in \mathbb{B}$.

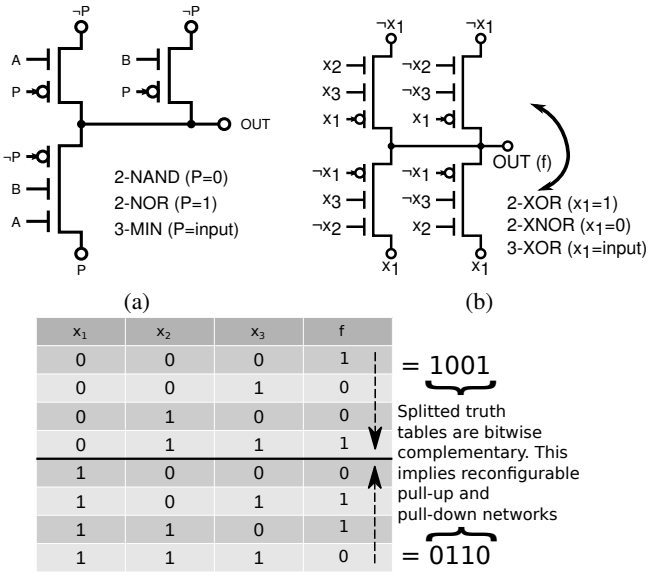


Fig. 3: (a) The transistor-level schematics of the reconfigurable logic gates (a) MINORITY and (b) XOR3, as presented in [23], [19]: (c) The functionality of the gates changes with the value applied to P and x_1, respectively.

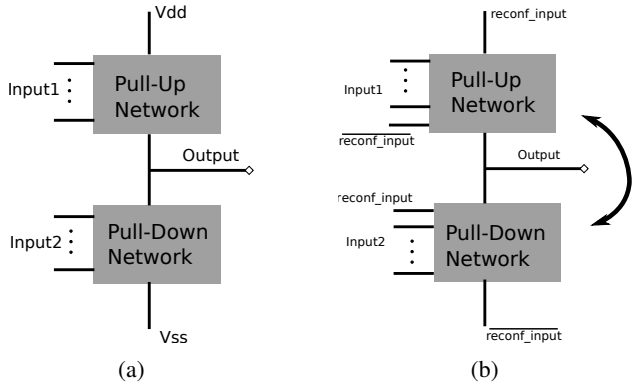


Fig. 4: (a) Fixed pull-up and pull-down networks in case of complementary MOS logic gates. (b) Switchable pull-up and pull-down networks in case of RFET-based logic gates [6]

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

Proof. For a self-dual function, because of Eq. 1, only half of inputs are sufficient to completely specify the function. From this, only 2^{n-1} combinations for n inputs exist. Hence, the total number of self-dual functions for n inputs considering both polarities (0 and 1) is equal to $2^{2^{n-1}}$ which is the square root of the total number of functions possible with n variables. □

Fig. 3c shows the self-dual 3-input function of the XOR3 logic gate. If the truth table is split by the value of x_1 (or any other arbitrarily selected literal), the two halves of the *XOR3* truth-table (*XOR2* and *XNOR2* if split by x_1) are dual to each other.

In [6], the authors showed that self-dual functions are a logical abstraction for ambipolar nanotechnologies. The multiple functionalities exhibited by RFET-based logic gates as shown in Figure 3 are due to the switchable pull-up and pull-down networks, as shown in Fig. 4. The switching of polarities of individual transistors in their respective pull-up and pull-down networks is caused by the change of the potential at the program gate, as shown in Fig. 3a and 3b. This change of the potential causes PFETs to become NFETs and vice-versa, which switches the pull-up and pull-down networks, as shown in Fig. 4b. This corresponding switch in electrical behavior is abstracted conveniently by a self-dual function. Only with self-dual functions, the polarity switch in individual transistors creates a conducting path between the output and the source (or drain) leading to the realization of the dual of the original function.

D. Terminologies

We introduce some terminologies, which will be used throughout the rest of the paper. A given circuit is represented as a direct acyclic graph consisting of nodes and edges. Nodes are data structures representing logic gates as defined by a given logic representation (AIG, XMG, etc.). Edges denote the connections between nodes. Without losing generality, the terms *circuit*, *logic graph*, *logic network* are used interchangeably throughout the manuscript.

1) *Self-duality density*: We define the term *self-duality density* for a circuit (or a logic network) as the ratio of the 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 the total number of functions) compared to the non-self-dual functions. Moreover, among two-input functions, self-duality exists in those functions which are equivalent to either the inputs or to their complements (for example, $f(a,b) = a$ or $f(a,b) = \overline{a}$). Such functions are implemented in circuits as wires (or use an inversion) and hence their implementation in RFETs is identical to that of CMOS. Thus, two-input self-dual functions are referred to as *trivial functions*. For self-dual functions with more than two inputs, their implementation with RFETs requires fewer transistors compared to their CMOS counterpart [19], [6]. These functions will have a direct impact on the area of the circuit. Hence, self-dual functions with 3 or more inputs are called *nontrivial functions*.

E. Majority logic synthesis

In this section, we review majority logic synthesis. The majority function $\langle x_1, x_2, x_3 \rangle$ of three Boolean variables x_1, x_2, x_3 evaluates to true if and only if at least two of the three variables

have a value of true. The majority function can be expressed in disjunctive normal form or conjunctive normal form, i.e.,

$$\begin{aligned}\langle x_1, x_2, x_3\rangle &= x_1x_2 + x_1x_3 + x_2x_3 \\ &= (x_1 + x_2)(x_1 + x_3)(x_2 + x_3). \end{aligned} \quad (3)$$

By setting one of the three Boolean variables x_1, x_2, x_3 in the majority function $\langle x_1, x_2, x_3\rangle$ to a constant value 0 or 1, one obtains the logic functions AND and OR, respectively, i.e.,

$$\langle 0, x_1, x_2\rangle = x_1x_2 \text{ and } \langle 1, x_1, x_2\rangle = x_1 + x_2, \quad (4)$$

Equation 4 is often called the *containment rule* of majority.

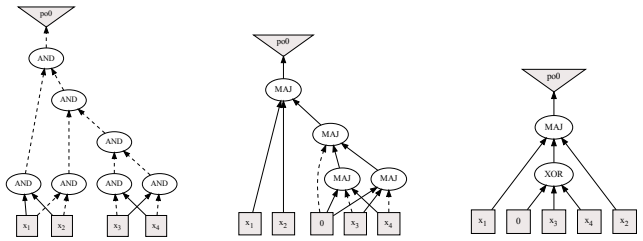
F. Earlier work on XMG

The authors of [10] proposed a logic representation, called *XOR-MAJ Graph* (XMG), consisting of three-fanin majority (MAJ) gates, three-fanin XOR gates, and inversion. The representation enables a size-proportional representation of both, n-input unate and n-input binate logic functions. XMGs were first introduced in [10] as a means for the underlying logic representation for exact synthesis. Exact synthesis solves the problem of finding an optimum network for a given function. Since exact synthesis uses SAT solving or enumeration, the runtime of exact synthesis tools directly depends on the size of the logic representation. Since XMGs have both binate (*XOR*) and unate (*MAJ*) nodes, this results in a size-proportional logic representation for both n-input unate and n-input binate functions compared to the representation of binate logic (*XOR*-based logic) of MIGs or AIGs [11]. Algebraic optimizations for XMG-based logic synthesis were proposed in [11]. The authors explored Boolean algebraic optimizations for *XOR* and *XOR-MAJ* logic and were able to achieve depth optimizations.

G. Classification of Boolean functions

Two Boolean functions $f(x)$ and $g(x)$ over the variables $x = x_1, \ldots, x_n$ belong to the same class C of functions if they are equivalent modulo some fixed set T of function transformations. In other words, if f can be transformed into g (or vice versa) by applying a sequence of transformations from T, then f and g are T-equivalent. The three most common function classes are (1) P: permutation of inputs; (2) NP: negation of inputs and permutation of inputs; (3) NPN: negation of inputs, permutation of inputs, and negation of outputs. These classes play an important role in technology-independent logic synthesis since two functions belonging to the same class can be represented with the same graph structure modulo the respective input and output transformations [37]. For example, the functions $f_1 = x_1x_2 + x_3$ and $f_2 = x_1 + x_2x_3$ are NPN-equivalent because by swapping the variables x_1 and x_3 in f_1 the function f_2 is obtained. Hence, if a node-minimum AIG for f_1 is known, then a node-minimum AIG for f_2 can be derived by swapping the inputs x_1 and x_3.

In the following, we use function classification (or function *canonization*) to perform Boolean matching in logic synthesis and technology mapping techniques. Boolean rewriting requires a database of size-minimum circuits for all Boolean functions. With the help of function classification [38], the database can be reduced to one size-minimum circuit per class. In technology mapping, the pre-enumeration and hashing of the NP-equivalent functions of all cells in the technology library enables us to match Boolean functions with cells more quickly [12].



(a) AIG (b) MIG (c) XMG

Fig. 5: Different logic representations of the function, $f = \langle x_1, x_2, (x_3 \oplus x_4)\rangle$. One can notice that the number of nodes and edges are the lowest in the XMG representation.

IV. Preserving self-duality

A. During logic synthesis

Due to their reconfigurability at the device level, RFETs enable efficient implementations of nontrivial self-dual logic functions in terms of the number of transistors [6] compared to CMOS. For example, a XOR logic gate with 3 inputs (shown in Fig. 4) requires $4 + 2$ (for P and P′) transistors when using RFETs as compared to 22 transistors when realized in CMOS technology [19]. This implies that circuit implementations with RFETs lead to area reductions if they have a high density of nontrivial (3 or more input functions) self-dual gates. In order to use this property, it is therefore imperative that the self-duality in a logic representation is preserved through logic optimizations. From a logic representation perspective, if we consider AIGs (consisting of two-input AND gates with complement-edge attributes), a nontrivial self-dual function is decomposed into multiple AIG nodes. Similarly, for MIGs, parity-based self-dual functions cannot be represented in a compact manner using *MAJ* nodes alone [11]. Various logic optimization techniques using cut-based techniques on AIGs and MIGs can allocate different cuts for the decomposed self-dual logic, thereby losing self-duality.

In contrast, XMGs use *XOR* and *MAJ* nodes as logic primitives. Each MAJ and odd-input XOR function is self-dual, so using XMGs can better preserve self-duality during logic optimization compared to other logic representations. This can easily be seen in Fig. 5. The figure shows three logic representations of the same function $f = \langle x_1, x_2, (x_3 \oplus x_4)\rangle$. The AIG logic representation requires 7 gates, while the same function has 4 and 2 gates when represented as MIG and XMG, respectively. In the example, there are more edges in the AIG and MIG representations than in the XMG representation. This leads to an increase in the number of competing structural cuts

Algorithm 1: Boolean filtering and resubstitution

Data: Window W in a logic network with root node n
Result: Node resubstitute for n or $\perp$ if no resubstitution has been found

Set $M \leftarrow \mathrm{W.computeMFFC}(n)$;
Set $D \leftarrow \mathrm{W.collectDivisors}(n) \backslash M$;
Set $TT \leftarrow \mathrm{W.simulate}()$;
$\mathrm{sortByDBP}(D, TT, n)$;
for $i \leftarrow 0$ **to** $|D|$ **do**
  if $3 \cdot \mathrm{DBP}(D[i]) < \mathrm{DBP}(n)$ **then**
    return $\perp$;
  for $j \leftarrow i+1$ **to** $|D|$ **do**
    if $\mathrm{DBP}(D[i]) + 2 \cdot \mathrm{DBP}(D[j]) < \mathrm{DBP}(n)$ **then**
      break;
    for $k \leftarrow j+1$ **to** $|D|$ **do**
      if $TT[n] = TT[i] \oplus TT[j] \oplus TT[k]$ **then**
        return $W.\mathrm{xor3\_resub}(n, D[i], D[j], D[k])$;
      if $TT[n] = \neg TT[i] \oplus TT[j] \oplus TT[k]$ **then**
        return $W.\mathrm{xor3\_resub}(n, \overline{D[i]}, D[j], D[k])$;
return $\perp$;

of the logic network in logic optimization and technology mapping phase. With the above benefits in mind, we have developed an XMG-based logic optimization flow that addresses these issues and helps to achieve area reductions for RFET-based circuits.

B. During versatile technology mapping

One of the limitations of the previous work [13] is the absence of a technology mapper that can map with arbitrary logic representations. The authors in [13] used XMG as the graph representation and carried out logic optimization intending to preserve self-duality. However, technology mapping was performed using ABC's native technology mapper, where AIG is the default logic representation. Thus, the XMG graph representation has to be converted to an AIG graph representation. During this process, individual XMG nodes are represented with multiple AIG nodes. This conversion leads to an increase in the number of competing cuts for large self-dual logic nodes. This can be understood from the Fig. 5 where the self-dual nodes of MAJ are represented using multiple AIG nodes. This increase in the number of competing cuts during mapping can disrupt the self-duality density of the circuit leading to suboptimal results in the context of area reduction for RFETs-based circuits. In our experiments, we found that in comparison to the native AIG-based technology mapping in ABC, this XMG-based logic graph leads to a tripling of the number of competing cuts during technology mapping within ABC. As a result, several optimal cuts that can preserve self-duality are lost during the technology mapping phase. Therefore, a logic-representative agnostic technology mapping is essential for our work. We have explored this observation and the explanation is carried out in Section VIII-D4.

V. Advanced logic synthesis techniques

While the prior work [10], [11] introduced XMGs and algebraic optimizations for them, we are extending the repertoire of Boolean methods with a resubstitution and NPN-based cut-rewriting technique.

A. XMG resubstitution

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, [38]).

Resubstitution algorithms are available for different multi-level logic representations including AIGs [39], [38], MIGs [40], [41], and logic networks [42], [43], [44] focusing on two-input AND operations, three-input MAJ operations, and combinations of two-input gates such as XOR-ANDs, AND-XORs, or three and two-input gates such as MUX-XORs, respectively.

Computing three-input XOR 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

$$f_n(x) = f_{d_1}(x) \oplus f_{d_2}(x) \oplus f_{d_3}(x) \tag{5}$$

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 Equation 5 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 [45]: a resubstitution of a target node n with function $f(x)$ and divisor 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

$$f(\hat{x}_i) \neq f(\hat{x}_j) \implies \bigvee_{1 \leq a,b \leq 3, a \neq b} d_a(\hat{x}_i) \neq d_b(\hat{x}_j). \tag{6}$$

Utilizing Equation 6, 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 $f_n(\hat{x}_i) \neq f_n(\hat{x}_j)$, 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 input assignments for which $f_n(\hat{x}_i) \neq f_n(\hat{x}_j)$ and $f_d(\hat{x}_i) \neq f_d(\hat{x}_j)$.

Example V.1. Suppose that n is a node to be substituted and $D = \{d_1, d_2, d_3, d_4\}$ are divisors with the following truth tables:

Node	TT	DBP
n	1001 0110	16
d_1	0000 0100	4
d_2	0111 1111	4
d_3	1000 0000	4
d_4	1111 1011	4

The absolute distinguishing bit power $\mathrm{DBP}(n) = 16$, whereas the relative distinguishing bit powers $\mathrm{DBP}_n(d_i) = 4$ for $i \in \{1, \ldots, 4\}$. We use a counting argument as a necessary condition to conclude that no Boolean operation using three (or less) of the given divisor functions is sufficient to synthesize n. Assuming that the given divisor functions distinguish different bit pairs, any subset of D of size 3 can distinguish at most 12 bit pairs. However, n requires 16 bit pairs to be distinguished. In other words, regardless of which three divisors one picks, there is always at least one bit pair that cannot be distinguished. This can be easily verified by looking at the truth tables of the divisor functions: the two least-significant bits of all divisor functions are equal, but the two least-significant bits of n are not.

Algorithm 1 shows our Boolean filtering and resubstitution algorithm as pseudocode. The algorithm first computes the MFFC (line 1), collects the divisors in W (line 2), which are all nodes excluding the MFFC, and simulates the nodes bottom-up such that each node n in W has a corresponding truth table $TT[n]$ (line 3). 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 a divisor triple that passes the test in Equation 5 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 Equation 5 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). Finally, if the algorithm reaches line 12, a divisor triple d_i, d_j, d_k has been found that passes all filtering checks. The algorithm then creates a new XOR node $d_i \oplus d_j \oplus d_k$ and substitutes n with it if and only if the corresponding truth tables $TT[n]$ and $TT[i] \oplus TT[j] \oplus TT[k]$ are equal (line $12-13$). This test is performed twice (line $14-15$)—once for each polarity of the first divisor.

If $|M| > 1$, the proposed resubstitution algorithm reduces the number of nodes in the logic representation by $|M| - 1$. If $|M| = 1$, the algorithm replaces a MAJ node by an XOR node.

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 [38] or compute size-optimum subnetworks on-the-fly using exact synthesis [46], [47]. DAG-aware rewriting [38], fast cut enumeration techniques [48], NPN canonization [37] of Boolean functions, and efficient caching [47] enable scalability.

Rewriting XMGs has been first proposed in [10] 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 us to limit the maximum synthesis effort per function. We consider exact XMG rewriting for three different sets of gate primitives:

1) Three-input MAJ gates with two-input XOR gates as originally proposed by [10];
2) Three-input MAJ 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 MAJ gates without constants and three-input XOR gates to improve the internal self-duality of a logic network during rewriting.

In practice, when mapped to RFETs, the best optimizations is achieved with the first set of gate primitives. The MAJ gate and the three-input XOR gates are large primitives and hence, fine granular optimizations can be lost. This is an inverse scenario that is explained in Section IV-B. Since the individual nodes are large, certain competing cuts (that could have been generated using smaller logic representation) required for optimal area reduction are lost. Also, for circuits, which do not have high self-duality density, the optimizations can be suboptimal and hence, the obtained graph representation can have a much higher number of edges (due to the undue presence of constants) than possible with smaller gate primitives.

VI. Logic representation-agnostic mapping

A. Versatile Mapper

In design automation, mapping is the process of expressing a logic graph using a set of primitives. Commonly, mapping either refers to LUT mapping or standard-cell mapping [49]. In the present work, we focus on standard-cell mapping in the context of emerging nanotechnologies. Once all the logic optimizations are carried out over a logic graph, the logic graph is converted to a k-bounded network, called subject graph. During technology mapping, the subject graph needs to be expressed using the standard cells present in a given cell library. This matching is often carried through Boolean matching [50].

Our versatile mapper supports arbitrary graph representations, such as AIG, XMG, XAG or MIG, to represent the subject graph in standard cells (as given in a technology library). In our work, the technology library is an RFET-centric generic library (`.genlib`) consisting of logic gates as proposed in [19]. The mapping can be optimized for either area or delay but in the present work, we focus only on area optimization. Our mapper follows four main steps:

1) *Library generation:* In the library generation phase, our mapper generates a hash table for the gate primitives listed in the technology library. The hash table consists of NP enumerations for each of the gate primitives subject to filtering of enumerations that are functionally symmetric [49]. We maintain a data structure that contains the delay and area values of each gate configuration.
2) *Cut enumeration:* In this step, we traverse the logic network in topological order and enumerate cuts with up to k inputs. For each of the cuts, truth tables are computed which are later used during Boolean matching to find a match in the hash table of the gate primitives. The cut enumeration technique used in our mapper is based on priority cuts as suggested in [51], [52].
3) *Boolean matching:* In this step, the truth table for each cut of the subject graph is looked up in the hash table (generated in the first step) to select gates that can implement it. We consider both polarities of the cut during matching to enable logic sharing of inverters and to avoid additional inverter delay.
4) *Optimization objective:* Once the matching is done for each of the cuts, a *cover* is selected. A *cover* is a set of cuts so that all cuts in the set are either rooted at the primary output or at the leaves of another cut. The cover is selected so that an optimum area or delay for the circuit is achieved. During delay minimization, the primary objective is to have the smallest delay of the largest path of the cover and during area minimization, the area of the cover is minimized. Various heuristics [53], [51] are followed during this step to attain the best possible mapping.

For more details on actual implementation of our mapper, readers are requested to refer to [14].

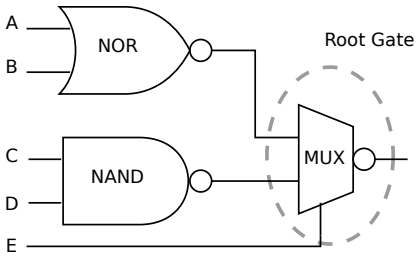


Fig. 6: Supergate generation. Here, MUX is the root gate with NANDs and NORs as the new input pins. The supergate thus generated has five inputs.

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 + 65280	65536	7 + 215	222

B. Support of supergates

Using supergates is an efficient technique as suggested by Chatterjee et al. to mitigate structural bias [54]. Structural bias arises from the fact that the structure of the starting logic graph representation dictates the final mapping quality to a large extent. By combining several gate primitives from the cell library, a list of supergates is precomputed to be used later during the technology-independent mapping step. Supergates aim to explore unique combinations of gate primitives which otherwise cannot be used during technology mapping [54]. An example is shown in Figure 6. Here, *MUX* is the root gate, and its two inputs are connected to outputs of two other logic gates. Consequently, a 5-input supergate is realized. Similarly, various other combinational supergates are precomputed and added to the hash table so that they are available for matching during technology-independent mapping. From the previous section, an obvious outcome is the increase in the size of the hash table created during the library generation stage. Supergates lead to improved quality of mapping at the expense of requiring additional runtime [12].

Our implementation can read supergate libraries produced by the open-source tool ABC [15]. For each entry in a `.super` file, our implementation computes the truth table, the area, and the delay. We then add it to the hash table generated in the library generation step. Once added to the hash table, the supergates are available to the mapper during the Boolean matching step.

VII. Creating self-dual benchmarks

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

Algorithm 2: Generate self-dual XMG network

Data: num_pis, $levels$, $nodes\_per\_level$, $index$
Result: XMG network N

1 Set $signalList \leftarrow []$;
2 Set $counter \leftarrow 0$;
3 **for** $k \leftarrow 0$ to *num_pis* **do**
4 | $signalList$.add(N.create_pi());
5 **for** $i \leftarrow 0$ to *levels* **do**
6 | **for** $j \leftarrow 0$ to *nodes_per_levels* **do**
7 | | $fanins \leftarrow signalList.randSubSet()$;
8 | | **if** $counter < index$ **then**
9 | | | $node \leftarrow N$.create_selfdual_gate($fanins$);
10 | | **else**
11 | | | $node \leftarrow N$.create_normal_gate($fanins$);
12 | | $signalList$.add(node);
13 | | $counter \leftarrow (counter + 1) \mod 10$;
14 **for** $o \in signalList.not\_used()$ **do**
15 | N.create_po(o);
16 **return** N

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.

Hence, in order to evaluate the efficacy of our approach compared to state-of-the-art logic synthesis approaches for RFET-based standard-cell mapping, we adopt a simple graph-based technique to generate benchmark circuits with varying numbers of self-dual logic gates. These benchmarks are built in a level-by-level fashion from the primary inputs to primary outputs. There have been multiple previous works targeting benchmark generation [55], [56].

We use Algorithm 2 to generate benchmarks with different self-dual densities, starting from an empty XMG network. The algorithm takes following four parameters as inputs: *the number of Primary Inputs (PIs)* (num_pis), *the number of levels* ($levels$), *the number of nodes per level* ($nodes\_per\_level$) and *the self-duality index* ($index$).

For a given set of the above three parameters– num_pis, $levels$, $nodes\_per\_level$, we generate 10 different benchmarks by assigning values $\{1 \rightarrow 10\}$ sequentially to the self-duality index ($index$). A self-duality index value v implies that out of every 10 nodes added, v are self-dual nodes (MAJ or 3-input XOR) and $10 - v$ are normal nodes (AND, OR, 2-input XOR) (line $8-11$). For example – let us consider the following values for the parameters (of Algorithm 2) – $num\_pis = 12$, $levels = 512$, $nodes\_per\_level = 131$. In this case, the Algorithm 2 creates 10 different circuits, all with 12 primary inputs, 512 levels between the primary inputs and primary outputs, and 131 maximum nodes in each level. Each benchmark is guided with the self-duality index value ($index$) chosen sequentially from $\{1 \rightarrow 10\}$ so that the 10 benchmarks have varying levels of self-duality density. The algorithm maintains a list of signals to keep track of all generated nodes (line 12). The algorithm first generates the primary inputs of the XMG network and adds them to the list (line $3-4$). It then adds new gates in a level-by-level fashion by randomly selecting fanins from the updated signal list (line 7). It is to be noted that self-duality index value of v (let's say 5) does not correspond to $(10 \times v)\%$ (50%) of self-duality density. This is primarily because during the construction of the circuit, nodes are added only after checking whether another node with the same fanins already exists in the graph or not. In this way, XMG optimizes away some of the redundant nodes. Finally, those nodes that are never referenced by any other node are marked as primary outputs (line $14-15$). The source code of the benchmark generator is available online[1].

VIII. Experiments

In this section, we describe our experimental setup and discuss the results obtained. All algorithms have been implemented in the open-source logic synthesis tool *mockturtle* from the EPFL logic synthesis libraries [16]. For technology mapping, we use the RFET-based generic library consisting of logic gates as mentioned in [19].

A. XMG-based Flow

Our proposed XMG-based flow comprises a logic synthesis and a technology mapping flow, where we use XMGs as the graph representation for a given circuit. We carry out the following steps in this flow:

1) Starting with the graph representation of a given circuit, we first convert it to a 4-input LUT-based logic graph. The procedure is similar as suggested in [57], [58] albeit with an additional step of converting the LUT-network into an XMG network. We then resynthesize[2] each 4-input LUT of the logic graph into the equivalent XMG representation by using a pre-generated 4-input NPN class.
2) We then apply the proposed two algorithms (resubstitution and 4-input NPN-based rewriting) iteratively to the obtained XMG until no size improvement of more than 0.5% is possible
3) Finally we use the versatile technology mapper on the obtained XMG from step 2. We use the RFET-based generic library and compute the supergates within our mapper to achieve an area-oriented mapping.

B. Experimental Setup

We carry out experiments on two sets of benchmarks. First, we use the synthetic benchmarks (as described in Section VII) and compare the post-mapping areas for different flows. This gives an empirical evidence of our proposed XMG-based synthesis flows. For post-mapping area, we use the following synthesis flows–

[1] https://github.com/shubhamrai26/self-dual-experiments
[2] Calling *node resynthesis* technique as defined in mockturtle framework [16]

TABLE II: Post-mapping area comparison for synthetic self-dual benchmarks using different synthesis flows.

SD-index	SD-density (%)	baseline abc_rwrs (# of transistors)	mtl_AIG (%)	abc_XMG (%)	Ours (%)
1	6.63	29904.22	0.93	-0.13	1.15
2	14.97	34248.61	0.79	-0.06	1.07
3	14.97	37732.53	0.62	-0.07	0.92
4	29.61	41296.80	0.57	-0.02	0.94
5	29.85	44717.89	0.62	0.05	1.03
6	40.27	48058.86	0.59	0.08	1.07
7	53.30	51154.18	0.58	0.12	1.11
8	62.79	54031.58	0.57	0.20	1.14
9	73.58	56710.74	0.58	0.32	1.17
10	100.00	61111.34	0.50	0.52	1.13

1) **abc_rwrs:** Native ABC flow is used to carry out both logic optimization and technology mapping. Here, AIG is the used graph representation. This flow is the *baseline* in our experiments.
2) **mtl_AIG:** AIG is the used graph representation. We use the logic-representation agnostic technology mapper within the mockturtle framework to compute post-mapping area.
3) **abc_XMG:** The flow as mentioned in [13] is used. Here, we use *mockturtle* for XMG-based logic optimizations and then carry out technology mapping with ABC.
4) **Ours:** This is the proposed XMG-based flow.

Then, we evaluate our proposed approach using real benchmarks in the form of cryptographic benchmarks [17], [18]. We carry out detailed analysis of our individual contributions. The area comparison is carried out in terms of number of transistors. Since we are comparing different logic synthesis flows, a reduction in the number of transistors has a direct impact on reducing the overall area of the RFET-based chip. This is the normalized area as mentioned in the `.genlib`. The three flows (mtl_AIG, abc_XMG, and ours) are compared with the baseline flow in terms of percentage. A negative percentage implies that the baseline flow is better than the other flows while positive percentages imply these flows are better than the baseline.

Throughout the experiments, we calculate *self-duality density* for the flows which use XMGs, as follows:

$$\text{sd-density} = \frac{\text{\# of 3-XOR nodes} + \text{\# of MAJ nodes}}{\text{total \# of nodes}} \tag{7}$$

where, 3-XOR and MAJ nodes are only those nodes in the network that do not have any inputs which are either $constant-0$ or $constant-1$. It is used to indicate the amount of self-duality that exists in the graph representation of the circuit.

C. Synthetic self-dual benchmarks

We use Algorithm 2 to generate 100 sets of benchmarks. For each set, the self-duality index in Algorithm 2 is iterated from 1 to 10 to generate 10 benchmarks for a particular set of parameters. Hence, in total, we have 1000 benchmarks to evaluate. The value of other parameters are chosen randomly for the 100 sets as mentioned in Section VII.

Once the benchmarks are generated, we compare the post-mapping area for the above-mentioned synthesis flows. TABLE II shows the comparison of the post-mapping area using synthetic benchmarks. We calculate the mean value over the entire benchmark set in the following way. For a particular value of the self-duality index, we calculate the mean over the 100 benchmarks generated. For example, the second column shows the mean of self-duality density calculated for each of the 100 benchmark sets corresponding to a given particular value of the self-duality index. The first two columns show the benchmark's self-duality index (*index*) and the corresponding self-duality density after logic optimizations step.

The next four columns in TABLE II show the *mean* of post-mapping area over 100 benchmarks for a particular value of self-duality index for four different flows. The column *baseline* shows the mean post-mapping area for the ABC native flow in terms of number of transistors. The next three columns show area comparison with respect to the baseline results in percentage. We can see that as compared to the baseline, the *mtl_AIG* columns achieve better area results. This can be ascertained due to better mapping results with the technology mapper within the mockturtle framework [14].

For the XMG-based flow using the ABC technology mapper, we get better results only with higher self-duality ratios. This is consistent with our assumption that with higher self-duality in circuits, the XMG-based flow achieves better area results. The last column shows area results with our proposed approach. The last column of the table shows that our versatile mapper achieves better area results than the ABC's technology mapper for XMG-based flow. We acknowledge that the improvement with XMG-based flow, though consistent, is close to $\sim 1\%$ only and is not so significant. This can be ascertained due to the fact that synthetic benchmarks generated using simple graph-based techniques are devoid of irregularities which are present in an actual benchmark [56]. Further, the AIG-based flows are well-established flows and hence, they can also achieve relatively good optimization. Due to these limitations, we focus our evaluation on real cryptography benchmarks.

D. Cryptographic benchmark suite

While the previous experiment was conducted on synthetic benchmarks, we next present an evaluation on cryptographic benchmarks. These benchmarks were taken from high-level cryptography protocols such as *Fully Homomorphic Encryption* (FHE) and secure *Multi-Party Communication* (MPC) [17], [18][3]. This 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). We have not considered the EPFL benchmark suite as the benchmarks are not representative of our use case. Almost all the benchmarks have poor self-duality density, except for a few benchmarks such as *square*.

*1) **Runtime improvement with filtering in XMG resubstitution**:* In order to measure the improvement in runtime using

[3]https://homes.esat.kuleuven.be/nsmart/MPC/

TABLE III: Runtime improvement in XMG resubstitution using the proposed filtering rule

Benchmarks	Size before	Runtime w/o filter (s)	Runtime w/ filter (s)	Improv. (%)
AES-expanded	25 435	20.19	5.30	73.75
AES-non-expanded	31 642	26.16	6.79	74.04
DES-expanded	20 199	447.56	37.85	91.54
DES-non-expanded	20 177	443.11	35.03	92.09
adder_32bit	99	0.01	0.01	0.00
adder_64bit	203	0.03	0.02	33.33
comparator_32_s_lt	119	0.03	0.02	33.33
comparator_32_s_lt	129	0.05	0.02	60.00
comparator_32_uns_lt	119	0.04	0.01	75.00
comparator_32_uns_lt	129	0.06	0.02	66.67
md5	27 867	11.31	4.34	61.63
mult_32x32	5378	4.30	0.84	80.47
sha-1	39 426	8.96	4.78	46.65
sha-256	83 381	22.84	12.48	45.36
Average				59.48

TABLE IV: Improvement of self-duality ratio after calling only resub, rewrite followed by resub. All the algorithms are called until convergence

Benchmarks	init (%)	rs (%)	rw;rs (%)
AES-expanded	4.68	19.02	19.07
AES-non-expanded	4.87	17.69	17.70
DES-expanded	29.79	34.04	34.06
DES-non-expanded	29.60	34.13	34.11
adder_32bit	31.31	96.88	96.88
adder_64bit	31.03	98.44	98.44
adder_128bit	25.25	67.55	67.55
comparator_32_s_lt	38.66	41.03	41.03
comparator_32_s_lteq	38.10	40.32	40.32
comparator_32_uns_lt	38.66	41.03	41.03
comparator_32_uns_lteq	38.10	40.32	40.32
md5	23.78	55.43	55.41
mult_32x32	28.77	41.07	41.33
sha-1	21.53	62.97	62.98
sha-256	26.87	69.86	69.83
Average	27.40	50.65	50.67

the XOR3-based filtering rule, we perform one iteration of resubstitution (with and without filtering) over cryptographic benchmarks using XMGs. The third and the fourth column in TABLE III show the runtime for our resubsitution algorithm for both flows. One can see that we achieve on average a runtime improvement of 59.48% across all benchmarks.

*2) **Improving self-duality density**:* We evaluate the impact of the proposed algorithms on the self-duality of the circuit. We follow a simple approach here. As done in all the experiments, we read the benchmarks in AIG and convert them to XMG using the *node resynthesis* technique. Then, we invoke algorithm(s) iteratively until no size improvement of more than 0.5% is possible. We then calculate the self-duality density for the logic graph. The results are shown in Table IV. The columns show the self-duality density for the logic graphs using different algorithms. The first column *init* shows the initial self-duality density calculated on the XMG logic graph of individual benchmarks before invoking the algorithms. The self-duality density does not change when invoking only the rewriting algorithm because the NPN-based rewriting will retain the number of MAJ and XOR nodes. However, as visible in the third column, resubstitution leads to a significant increase in the self-duality of individual circuits. This can be ascertained due to the addition of extra self-dual nodes (3-input MAJ and 3-input XORs) to the logic graph. The final column shows the self-duality density after calling the two algorithms sequentially. It can be seen that self-duality almost remains the same. This experiment demonstrates that the proposed logic synthesis algorithm helps to increase the self-duality of the circuit.

*3) **Area Comparison**:* Table V shows the post-mapping area comparison for cryptographic benchmarks with[4] and without supergates respectively. As in the case of synthetic benchmarks, here also we compare the post-mapping area for the four flows. For ease of readability, we have added the sd-density value for individual benchmarks from Table IV (last column). We show the improvement with respect to the baseline area results.

One can notice that most of the benchmarks from the cryptography domain have a high density of self-dual gates ($>50\%$), particularly parity functions as it is an integral logic function in any cryptographic applications. In case of benchmarks, where sd-ratio is lower ($<50\%$) (*DES*, *comparator*), the XMG-based approach gives poor results. Hence, both AIG-based approaches (*baseline* and *mtl_AIG*) achieve better area results. This is due to the fact that AIG representation consists of 2-input nodes and hence more smaller cuts are available for mapping compared to XMGs which consist of bigger XOR and MAJ nodes. Hence, the mapping achieves better optimization in terms of area-oriented mapping for these circuits. However, for benchmarks, *md5*, *SHA-1* and *SHA-256*, the proposed XMG-based approach (*Ours*) outshines other synthesis flows. and gives superior results. The higher self-duality in these circuits leads to the mapping of more self-dual logic gates thereby leading to more area reduction as explained in Section IV.

Consideration of supergates leads to even better results as it is able to mitigate structural bias issues [12]. However, in case of ABC-based flows (*baseline* and *abc_xmg*), the mapper leads to poor area results for some benchmarks. This can be ascertained as the mapper can get stuck in a local minimum as area-oriented mapping is an intractable problem and driven by heuristics. However, mockturtle-based flows (mtl_AIG and ours) are more consistent here as using supergates leads to more uniform area reduction (*comparator*, *multiplier*, *SHA-1*, *SHA-256*). Using supergates with our proposed approach achieves best area results of up to 12.36% for circuits with higher self-duality.

*4) **Exploring why higher self-duality density leads to better area results with XMG-based approach**:* In this set of experiments, we explore why XMG-based approaches lead to better area reduction as compared to AIG-based approaches. We use the mockturtle-based synthesis flows for this experiment (mtl_AIG and ours). We calculate three data– the total number

[4]using ABC command to create supergate library: `super -I 5 -L 3 -N 0 -T 1 -D 0.00 -A 0.00`. The command in ABC generates 5 input supergates by combining 3 levels of primitive gates. This command generates a total of $\sim$ 17000 supergates for mapping.

TABLE V: Comparison of post-mapping area without the use of supergates. Use of supergates lead to better area across all the flows particularly with mockturtle framework (mtl_AIG and Ours). High self-duality density leads to higher improvement over baseline for XMG-based flows

Benchmarks	Self-duality density (%)	Without supergates				With supergates			
		Baseline rwrs area (# of transistors)	xmg_rwrs (%)	mtl_aig (%)	Ours (%)	Baseline rwrs area	xmg_rwrs (%)	mtl_aig (%)	Ours (%)
AES-expanded	19.07	72231.00	-3.50	-4.98	-4.03	71568.00	-2.77	-1.34	-2.97
AES-non-expanded	17.7	92190.00	-1.29	-4.11	-1.26	90962.50	-1.32	-1.08	-1.26
DES-expanded	34.06	35419.00	-1.52	-8.44	-15.44	35322.00	-1.32	-2.03	-7.14
DES-non-expanded	34.11	35495.50	-0.94	-7.88	-15.75	35230.00	-1.27	-2.00	-7.72
adder_32bit	96.88	270.00	-10.74	0.00	0.00	270.00	0.00	0.00	0.00
adder_64bit	98.44	542.00	-11.25	0.00	0.00	542.00	0.00	0.00	0.00
comparator_32_s_lt	41.03	246.00	2.44	-22.97	-7.93	235.00	6.81	0.64	**6.17**
comparator_32_s_lteq	40.32	235.00	-4.47	-22.55	-27.23	225.50	-1.11	-8.87	-5.10
comparator_32_uns_lt	41.03	246.00	0.81	-22.97	-7.72	235.00	5.11	0.64	**6.60**
comparator_32_uns_lteq	40.32	235.00	-4.47	-22.55	-27.23	225.50	-1.11	-8.87	-5.10
md5	55.41	78489.00	2.20	5.81	**8.24**	79810.00	2.07	7.58	**9.93**
mult_32x32	41.33	9917.50	2.39	-7.34	-15.95	10208.00	0.74	-2.65	**3.92**
Sha-1	62.98	116425.00	1.17	7.72	**9.19**	118003.50	0.79	9.12	**10.61**
Sha-256	69.83	232028.50	-0.56	10.28	**9.55**	236238.00	0.84	12.08	**12.36**

TABLE VI: Comparison of ratios of self dual cuts, and gate area between AIG and XMG-based representation

Benchmark	AIG				XMG			
	total cuts	sd-cuts (%)	sd3-cuts (%)	sd-area-ratio (%)	total cuts	sd-cuts (%)	sd3-cuts (%)	sd-area-ratio (%)
AES-expanded	2906133	5.85	4.74	37.78	363829	24.50	18.92	37.82
AES-non-expanded	3754949	6.20	4.94	34.26	561621	29.12	23.40	35.93
DES-expanded	1109707	0.82	0.48	9.32	292045	8.07	4.92	24.97
DES-non-expanded	1117971	0.83	0.48	9.83	291256	8.08	4.92	25.74
adder_32bit	4591	3.81	1.31	97.96	127	96.06	47.24	97.96
adder_64bit	9576	3.76	1.29	98.99	255	98.04	48.63	98.99
comparator_32_s_lt	4084	1.15	0.44	35.76	564	14.54	4.96	45.80
comparator_32_s_lteq	3668	1.04	0.35	33.60	622	13.99	5.31	39.45
comparator_32_uns_lt	4084	1.15	0.44	35.76	556	14.93	5.04	45.56
comparator_32_uns_lteq	3668	1.04	0.35	33.60	622	13.99	5.31	39.45
md5	3395540	5.24	3.97	49.53	168548	37.75	27.01	66.71
mult_32x32	285268	3.01	2.12	34.32	39800	12.51	6.97	46.20
sha-1	5632714	6.62	5.38	67.23	174914	47.60	35.18	74.17
sha-256	11774568	6.22	5.02	76.18	370611	51.87	38.39	82.54
Average	**2002056.33**	**3.31**	**2.16**	**46.72**	**151104.07**	**35.80**	**20.72**	**54.38**

of cuts available during mapping, the number of total self-dual cuts, and the number of non-trivial self-dual cuts. These data give us an overall idea of what kind of cuts are available during mapping. Additionally, from post-mapping results, we compute the ratio of the area of self-dual logic gates to the total area of the circuit. Table VI show these values for both AIG and XMG-based flows. For each benchmark, we show the total cuts, the ratio of self-dual cuts to the total cuts, the ratio of non-trivial self-dual cuts to the total cuts, and finally the ratio of area contribution from self-dual logic gates to the total post-mapping area.

From Table VI, the number of total cuts in AIG is more than that in XMG as AIG uses a 2-fanin AND node compared to XMG that uses larger 3-fanin XOR and 3-fanin MAJ nodes. However, the ratio of self-dual cuts is much higher in the case of XMG as compared to AIG. An interesting observation is that this ratio is also higher for circuits that have a higher self-duality density (*sha-1*, *sha-256*, *md5*) which clearly indicates that XMGs offer more self-dual cuts during mapping compared to AIGs. Similarly, for circuits with higher self-duality density, XMG on an average has a higher *sd-area-ratio* (54.38% vs 46.72%) compared to AIGs. In fact, it can be noticed that XMG-based approach has higher *sd-area-ratio* for all the benchmarks. Due to the higher *sd-area-ratio*, XMGs lead to better area reduction for RFET-based implementation compared to AIGs for circuits with higher self-duality.

Another practical benefit with the XMG-based flow, that can be correlated with the results shown in Table VI and was earlier demonstrated in [10], is the improvement in the runtime for XMG-based flow. XMG-based flow has to iterate over fewer cuts as compared to the AIG-based flow which leads to reduction in the overall runtime of the synthesis flows. In terms of runtime to carry out technology-independent mapping for all the cryptography benchmarks, XMG takes in total 7.06 seconds as compared to AIG that takes 59.88 seconds.

IX. Summary and Conclusion

The present work explores both logic synthesis and technology mapping from an emerging technology perspective. With the particular aim of preserving self-duality in circuits, we investigate an XMG-based logic synthesis solution for RFETs-based circuits. XMGs are used for two reasons: (i) XMGs are a compact representation for both unate and binate logic; (ii) the logic primitives in XMGs (MAJs and XORs) can efficiently represent self-dual logic gates because both MAJ and odd-input

XORs are self-dual. Additionally, with our logic-representation-agnostic mapper, the limitation of previous work [13] of converting XMGs into AIGs before technology mapping has been resolved. In comprehensive experimental evaluations, we compare our XMG-based flow with three different sets of experiments and show that for circuits with higher self-duality, the XMG-based synthesis achieves better area results. Future work directions include the development of more optimization algorithms for XMGs. Additionally, measures to deal with the problem of structural bias within technology mapping needs to be explored.

ACKNOWLEDGMENTS

This research was supported in part by the German Research Foundation (DFG), project SecuReFET (project number: 439891087) and in part by the SNF grant "Supercool: Design methods and tools for superconducting electronics", 200021_1920981.

REFERENCES

[1] M. D. Marchi, D. Sacchetto, J. Zhang, S. Frache, P. E. Gaillardon, Y. Leblebici, and G. De Micheli, "Top-down fabrication of gate-all-around vertically stacked silicon nanowire fets with controllable polarity," *IEEE Transactions on Nanotechnology*, Nov 2014.

[2] A. Heinzig, T. Mikolajick, J. Trommer, D. Grimm, and W. M. Weber, "Dually active silicon nanowire transistors and circuits with equal electron and hole transport," *Nano Letters*, 2013, pMID: 23919720.

[3] M. H. Ben-Jamaa, K. Mohanram, and G. De Micheli, "An efficient gate library for ambipolar cntfet logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 2, pp. 242–255, Feb 2011.

[4] J. Trommer, A. Heinzig, T. Baldauf, S. Slesazeck, T. Mikolajick, and W. M. Weber, "Functionality-enhanced logic gate design enabled by symmetrical reconfigurable silicon nanowire transistors," *IEEE Trans. Nanotech.*, July 2015.

[5] M. Raitza, A. Kumar, M. Völp, D. Walter, J. Trommer, T. Mikolajick, and W. M. Weber, "Exploiting transistor-level reconfiguration to optimize combinational circuits," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017.

[6] S. Rai, M. Raitza, S. Sahoo, and A. Kumar, "Discern: Distilling standard-cells for emerging reconfigurable nanotechnologies," in *DATE*, 2020.

[7] A. Kuehlmann, M. K. Ganai, and V. Paruthi, "Circuit-based boolean reasoning," in *Proceedings of the 38th Annual Design Automation Conference*, New York, NY, USA, 2001, p. 232–237.

[8] L. Amarú, P. E. Gaillardon, and G. De Micheli, "Majority-Inverter Graph: A new paradigm for logic optimization," *TCAD*, 2016.

[9] I. Háleček, P. Fišer, and J. Schmidt, "Are XORs in logic synthesis really necessary?" in *DDECS*, 2017.

[10] W. Haaswijk, M. Soeken, L. Amarù, P. E. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *ASP-DAC*, 2017.

[11] Z. Chu, M. Soeken, Y. Xia, L. Wang, and G. De Micheli, "Structural rewriting in XOR-majority graphs," in *ASPDAC*, 2019.

[12] S. Chatterjee, *On algorithms for technology mapping (Doctoral Thesis)*. University of California, Berkeley, 2007.

[13] S. Rai, H. Riener, G. De Micheli, and A. Kumar, "Preserving self-duality during logic synthesis for emerging reconfigurable nanotechnologies," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 354–359.

[14] A. T. Calvino, H. Riener, S. Rai, A. Kumar, and G. De Micheli, "A versatile mapping approach for technology mapping and graph optimization," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 410–416.

[15] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *CAV*, 2010.

[16] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, and G. De Micheli, "The EPFL logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.

[17] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for mpc and fhe," in *EUROCRYPT*. Springer, 2015.

[18] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *ASCCCS*, 2017.

[19] S. Rai, J. Trommer, M. Raitza, T. Mikolajick, W. M. Weber, and A. Kumar, "Designing efficient circuits based on runtime-reconfigurable field-effect transistors," *TVLSI*, March 2019.

[20] L. G. Amaru, *New Data Structures and Algorithms for Logic Synthesis and Verification*. Springer, 2017.

[21] S. Rai, S. Srinivasa, P. Cadareanu, X. Yin, X. S. Hu, P.-E. Gaillardon, V. Narayanan, and A. Kumar, "Emerging reconfigurable nanotechnologies: Can they support future electronics?" in *ICCAD*, 2018.

[22] M. D. Marchi, D. Sacchetto, S. Frache, J. Zhang, P. E. Gaillardon, Y. Leblebici, and G. De Micheli, "Polarity control in double-gate, gate-all-around vertically stacked silicon nanowire fets," in *IEDM*, 2012.

[23] A. Heinzig, S. Slesazeck, F. Kreupl, T. Mikolajick, and W. M. Weber, "Reconfigurable silicon nanowire transistors," *Nano Letters*, 2012.

[24] J. Trommer, A. Heinzig, A. Heinrich, P. Jordan, M. Grube, S. Slesazeck, T. Mikolajick, and W. M. Weber, "Material prospects of reconfigurable transistor (rfets)–from silicon to germanium nanowires," *MRS*, 2014.

[25] Y. Lin, J. Appenzeller, J. Knoch, and P. Avouris, "High-performance carbon nanotube field-effect transistor with tunable polarities," *TNano.*, 2005.

[26] S. Tanachutiwat, J. U. Lee, W. Wang, and C. Y. Sung, "Reconfigurable multi-function logic based on graphene p-n junctions," in *DAC*, 2010.

[27] G. V. Resta, S. Sutar, Y. Balaji, D. Lin, P. Raghavan, I. Radu, F. Catthoor, A. Thean, P.-E. Gaillardon, and G. De Micheli, "Polarity control in WSe_2 double-gate transistors," *Scientific Reports*, 2016.

[28] T. Mikolajick, A. Heinzig, J. Trommer, T. Baldauf, and W. M. Weber, "The rfet—a reconfigurable nanowire transistor and its application to novel electronic circuits and systems," *Semiconductor Science and Technology*, 2017.

[29] M. Simon, A. Heinzig, J. Trommer, T. Baldauf, T. Mikolajick, and W. M. Weber, "Top-down technology for reconfigurable nanowire fets with symmetric on-currents," *IEEE Transactions on Nanotechnology*, Sept 2017.

[30] M. Simon, J. Trommer, B. Liang, D. Fischer, T. Baldauf, M. B. Khan, A. Heinzig, M. Knaut, Y. M. Georgiev, A. Erbe, J. W. Bartha, T. Mikolaick, and W. M. Weber, "A wired-and transistor: Polarity controllable fet with multiple inputs," in *2018 76th Device Research Conference (DRC)*, June 2018.

[31] J. Zhang, X. Tang, P. E. Gaillardon, and G. De Micheli, "Configurable circuits featuring dual-threshold-voltage design with three-independent-gate silicon nanowire fets," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Oct 2014.

[32] J. Trommer, A. Heinzig, T. Baldauf, T. Mikolajick, W. M. Weber, M. Raitza, and M. Völp, "Reconfigurable nanowire transistors with multiple independent gates for efficient and programmable combinational circuits," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 169–174.

[33] J. Nevoral, R. Růžička, and V. Šimek, "From ambipolarity to multifunctionality: Novel library of polymorphic gates using double-gate fets," in *DSD*, 2018.

[34] S. Rai, M. Raitza, and A. Kumar, "Technology mapping flow for emerging reconfigurable silicon nanowire transistors," in *DATE*, March 2018.

[35] J. Zhang, P. E. Gaillardon, and G. De Micheli, "Dual-threshold-voltage configurable circuits with three-independent-gate silicon nanowire fets," in *Proceedings - IEEE International Symposium on Circuits and Systems*, May 2013.

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

[37] Z. Huang, L. Wang, Y. Nasikovskiy, and A. Mishchenko, "Fast Boolean matching based on NPN classification," in *FPT*, 2013.

[38] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "DAG-aware AIG rewriting a fresh look at combinational logic synthesis," in *DAC*, 2006.

[39] A. Mishchenko and R. K. Brayton, "Scalable logic synthesis using a simple circuit structure," in *IWLS*, 2006.

[40] H. Riener, E. Testa, L. G. Amarù, M. Soeken, and G. De Micheli, "Size optimization of MIGs with an application to QCA and STMG technologies," in *NANOARCH*, 2018.

[41] H. Riener, E. Testa, W. Haaswijk, A. Mishchenko, L. G. Amarù, G. De Micheli, and M. Soeken, "Scalable generic logic synthesis: One approach to rule them all," in *DAC*, 2019.

[42] V. N. Kravets and P. Kudva, "Implicit enumeration of structural changes in circuit optimization," in *DAC*, 2004.

[43] A. Mishchenko, R. K. Brayton, J. R. Jiang, and S. Jang, "Scalable don't-care-based logic optimization and resynthesis," *ACM TRECTS.*, 2011.

[44] L. G. Amarù, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, J. Olson, R. K. Brayton, and G. De Micheli, "Improvements to Boolean resynthesis," in *DATE*, 2018.

[45] K. Chang, I. L. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," *TCAD*, 2008.

[46] 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 *DATE*, 2019.

[47] H. Riener, A. Mishchenko, and M. Soeken, "Exact DAG-aware rewriting," in *DATE*, 2020.

[48] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *ISFPGA*, 1999.

[49] G. De Micheli, *Synthesis and optimization of digital circuits*, 1st ed. McGraw-Hill, 1994.

[50] F. Mailhot and G. De Micheli, "Technology mapping using boolean matching and don't care sets," in *Proceedings of the Conference on European Design Automation*, ser. EURO-DAC '90. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990.

[51] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999, pp. 29–35.

[52] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2007, pp. 354–361.

[53] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in lut-based FPGA technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2331–2340, 2006.

[54] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping," in *ICCAD*, 2005.

[55] W. L. Neto, V. N. Possani, F. S. Marranghello, J. M. Matos, P.-E. Gaillardon, A. I. Reis, and R. P. Ribas, "Exact benchmark circuits for logic synthesis," *IEEE Design & Test*, vol. 37, no. 3, pp. 51–58, 2019.

[56] D. Stroobandt, P. Verplaetse, and J. van Campenhout, "Generating synthetic benchmark circuits for evaluating cad tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 9, pp. 1011–1022, 2000.

[57] A. Neutzling, J. M. Matos, A. Mishchenko, A. Reis, and R. P. Ribas, "Effective logic synthesis for threshold logic circuit design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 926–937, 2019.

[58] A. Neutzling, F. S. Marranghello, J. M. Matos, A. Reis, and R. P. Ribas, "maj- n logic synthesis for emerging technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 3, pp. 747–751, 2020.

Shubham Rai received the B.Engg. in electrical and electronic engineering and M.Sc. in Physics from Birla Institute of Technology and Science Pilani, India, in 2011. He is currently working towards the Ph.D. degree at Technische Universität, Dresden, Germany. His research focus is on design automation for reconfigurable nanotechnologies and their security implications.

Alessandro Tempia Calvin received the B.S. degree in Computer Engineering from the Politecnico di Torino, Turin, Italy, in 2017, and the M.S. degree in Computer Engineering from the Politecnico di Torino, in 2020, and Télécom Paris, Paris, France, in 2021. He is currently pursuing the Ph.D. degree in Computer Science with the Swiss Federal Institute of Technology Lausanne, Lausanne, Switzerland in the Integrated Systems Laboratory. His current research interests include design automation, logic synthesis, and emerging technologies.

Heinz Riener is a researcher at EPFL, Lausanne, Switzerland. He holds a Ph.D. degree in Computer Science from University of Bremen, Germany. He received his B.Sc. and M.Sc. degree from the Technical University Graz, Austria. From 2015 to 2017, he worked at the German Aerospace Center, Bremen, Germany, in the group of Avionics Systems. His research interests are logic synthesis, formal methods, and computer-aided verification of hardware and software systems.

Giovanni De Micheli is Professor and Director of the Integrated Systems Laboratory at EPFL, Lausanne, Switzerland. Previously, he was Professor of Electrical Engineering at Stanford University. Prof. De Micheli is a Fellow of ACM and IEEE, a member of the Academia Europaea and an International Honorary member of the American Academy of Arts and Sciences. His current research interests include several aspects of design technologies for integrated circuits and systems, such as synthesis for emerging technologies.

Akash Kumar (Senior Member, IEEE) is currently a Professor at Technische Universität Dresden (TUD), Germany, where he is directing the chair for Processor Design. He received the joint Ph.D. degree in electrical engineering in embedded systems from University of Technology (TUe), Eindhoven and National University of Singapore (NUS), in 2009. From 2009 to 2015, he was with the National University of Singapore, Singapore. His current research interests include design, analysis, and resource management of low-power and fault-tolerant embedded multiprocessor systems.