Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier xx.xxxx/ACCESS.xxxx.DOI

Shrinking FPGA Static Power via Machine Learning-Based Power Gating and Enhanced Routing

Zeinab Seifoori[1], Hossein Asadi[1](Senior Member, IEEE) and Mirjana Stojilović[2](Senior Member, IEEE)

[1]Sharif University of Technology, Tehran, Iran (e-mail: seifoori@ce.sharif.edu, asadi@sharif.edu)
[2]School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland (e-mail: mirjana.stojilovic@epfl.ch)

Corresponding author: Hossein Asadi (e-mail: asadi@sharif.edu).

ABSTRACT Despite FPGAs rapidly evolving to support the requirements of the most demanding emerging applications, their high static power consumption, concentrated within the routing resources, still presents a major hurdle for low-power applications. Augmenting the FPGAs with power-gating ability is a promising way to effectively address the power-consumption obstacle. However, the main challenge when implementing power gating is in choosing the clusters of resources in a way that would allow the most power-saving opportunities. In this paper, we take advantage of machine learning approaches, such as K-means clustering, to propose efficient algorithms for creating power-gating clusters of FPGA routing resources. In the first group of proposed algorithms, we employ K-means clustering and exploit the utilization pattern of routing resources. In the second group of algorithms, we enhance the power-gating efficiency by minimizing the power overhead introduced by power-gating logic and by taking into account the size of routing multiplexers, which influences the power-gating efficiency. Finally, we enhance and further develop the baseline FPGA routing algorithm to be aware and take advantage of power gating opportunities. The experimental results on Titan benchmark suite and the latest Intel Stratix-IV FPGA architecture in VTR 8.0 show that our approaches achieve an improvement of about 70%, on average, in reducing the FPGA static power consumption over the best power-gating approaches proposed in the previous studies.

INDEX TERMS Field-programmable gate arrays, static power consumption, power gating, routing algorithm, machine learning

I. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) have become an ubiquitous alternative to *Application-Specific Integrated Circuits* (ASICs), thanks to their compelling advantages such as reduced nonrecurring engineering costs, almost unlimited design flexibility, fast time-to-market, and inexpensive design updates. However, the penetration of FPGAs in the power-limited applications and devices (i.e., mobile phones) is lagging due to their relatively high static power consumption in comparison with ASICs [1]. Therefore, it is crucial to timely develop new techniques that could help shrink the FPGA static power consumption and enable them to compete with ASICs.

Previous research has shown that power gating can lead to significant power saving in the FPGA routing and logic resources [2]–[7]. These so-called power-gating regions can be controlled statically (using the FPGA configuration bits at configuration time) or dynamically (using an on-chip specialized circuitry at run-time). Given that the most significant part of the FPGA static power is consumed by nothing else but FPGA routing resources (70% up to 90% of the total static power consumption [8], [9]), it is the most advantageous to apply power-gating techniques to the FPGA routing resources.

Although power gating may seem to be a straightforward approach, the benefits of using it come with inevitable area and power overheads [10], [11]. Previous studies proposing fine-grained [3], [7], [12]–[16] or coarse-grained [6], [17] power-gating architectures, have mainly focused on granularity—determined by the number of resources in a power gating region—and employed relatively simple heuristics to find power-gating solutions [3]–[7], [16], [17]. These studies show that the probability of providing an optimal clustering through heuristic approaches is relatively low.

An entirely different approach to designing the power-gating regions is employing Machine Learning (ML) to enhance FPGA design automation. Several pioneering work attempted to enhance FPGA design-automation algorithms through optimizing their parameters with the help of ML approaches [18]–[22]. Furthermore, several studies employ ML techniques to improve the quality of congestion estimation in the placement and routing steps of the design automation process [23]–[26]. To the best of our knowledge, *none* of the previous studies have attempted to apply an ML-inspired approach to design power gating regions for the FPGA routing network in order to reduce the FPGA static power consumption.

In this paper, we introduce several new notions: *similarity metric*, *cluster pattern*, and *power gating efficiency*, with the goal of proposing a set of novel power-gating algorithms. These algorithms are all based on *K*-means clustering, a well-known ML approach that aims at grouping objects based on their similarity and can be adapted to the clustering problem at hand.

To validate the effectiveness of our proposed clustering algorithms, we implement them in *Verilog-to-Routing* (VTR) open-source toolset [27] and compare them with the previous work [5]–[7]. We carry out an extensive set of experiments using the industrial-scale Titan benchmarks [28], as well as the Intel Stratix-IV FPGA architecture model, the most advanced FPGA architecture with integrated support in the VTR 8.0 suite [27]. We exploit the latest version of *Circuit Optimization For FPGA Exploration* (COFFE) [29] supplied with 22nm technology model to extract the transistor sizing of the target FPGAs, and HSPICE circuit-level simulations for estimating the area and delay of the FPGA resources.

In this paper, we extend our previous work [30] with the following novel contributions:

- We propose a new clustering algorithm, which, for the first time, takes into account the size of the routing multiplexer as well as the overhead, in terms of area and the power-consumption, of the power-gating circuitry. The proposed algorithm aims to minimize the overall static power consumption of the FPGA routing network. We demonstrate that the proposed algorithm is significantly more advantageous than previous approaches, which are solely focused on maximizing the number of the candidate multiplexers that can be powered off.
- We design and implement a power-gating aware FPGA router to improve the utilization of the existing power-gating regions and further reduce the FPGA static-power consumption.

We show that employing our most-optimized routing algorithm, called, *SiM-IPR-MP*, with 32 clusters per switch matrix and the power-gating aware router can shrink the FPGA static power consumption by 53%, on average, whereas the best power-gating approach presented by Seifoori et al. [7] achieves 31% power reduction, on average; this presents an improvement of about 70% over the state of the art.

The remainder of the paper is structured as follows. First, we present the related work (Section II). Then, in Section III, we provide necessary background in FPGA routing architecture and *K*-means clustering, together with an example that motivates this work. In Section IV, we present an algorithm based on standard *K*-means clustering and our four proposed algorithms, namely, *SiM*, *SiM-PR*, *SiM-IPR*, and *SiM-IPR-MP*. In the same Section, we present the modified FPGA routing algorithm. In Section V, we focus on the experimental methodology and the results. Section VI concludes the paper.

II. RELATED WORK

In this section, we survey the related work from two perspectives: First, we review of the research studies on reducing the static power. Second, we review the research on applying machine learning approaches for increasing the efficiency of FPGA designs, for example by optimizing the area, timing, and power metric of FPGA designs through FPGA parameter tuning [18]–[22] or by ptimizing the routing and placement quality of FPGA designs [23]–[26].

A. POWER GATING APPROACHES

To reduce the FPGA power consumption, prior studies suggest applying the power gating techniques in either the FPGA logic or the routing resources. Here, we will focus on those studies which implement power gating in the routing resources, as they are the most related to our proposed approaches in this paper.

Two distinct power-gating techniques exist: *static power gating* (applied during the configuration time to power off the resources which will not be used) and *dynamic power gating* (applied during the runtime, to power off the FPGA modules during their idle periods of work). Our approaches, discussed later, fall into the former category.

In one of the first works, Bsoul et al. augment each routing *Switch Matrix* (SM) to operate in either the state of *always-ON*, *always-OFF*, or the state, which controlled dynamically by an on-chip controller module. This approach increases the routing cost of using resources outside the constrained area of the functional module to reduce the number of the always-ON SMs. The efficiency of this approach primarily depends on the number of entirely unused SMs, as the partially used SMs cannot be always powered off. The proposed approach decreases the power consumption by 70% to 84%. Even though the reported power saving is considerable, since the results are obtained based on the best number of *always-ON* SMs and also the experimental benchmarks are selected from MCNC benchmarks [31], the results cannot be generalized to commercial-like and more realistic benchmarks. In addition, implementing dynamic power gating is faced with great challenges such as a) controlling inrush current[1], and b) establishing a reasonable trade-off between the power overhead of

[1]The transient large current, which is flowed from power lines during the power state switching.

controller circuit and controlling signals and achieved power saving. Furthermore, extracting the resource idleness periods which should be long enough to justify resource overhead of power gating is challenging, especially in input-dependent and interactive applications. Finally, employing dynamic power gating is orthogonal to static power gating and our focus in this paper is enhancing the static power gating.

Li et al. proposed using a *Power-Control Hard Macro* (PCHM) to implement a coarse-grained power gating of the FPGA logic blocks and their associated connection blocks [17]. In addition, they manage to decrease the power consumption of the clock network, by embedding the clock gating logic into PCHM. Finally, they modify the cost function in the FPGA placement algorithm with the aim of minimizing the number of used power gating regions and hence increasing the power gating opportunities in each design. The proposed enhanced placement algorithm is orthogonal to our proposed enhanced routing algorithm.

Hoo et al. employed a static coarse-grained power gating approach [6]. In each SM, they created four power gating regions, composed of the unidirectional SM buffers of wires going in the same direction. In addition, they adapted the cost function in the FPGA routing algorithm to minimize the number of inactive power gating regions. In the proposed routing algorithm, the cost of employing each routing resource is scaled down exponentially with the number of nets through the power gating cluster associated with the routing resource. Hoo et al. also dynamically turn off the FPGA modules during their idle periods, to increase the power savings. They reported that approximately 40% power gating regions could be turned off in their experiments.

Gayasen et al. proposed a static power-gating scheme with various granularities composing of a rectangular array of *Configurable Logic Blocks* (CLBs). In addition, they propose the *Region Constraint Placement* (RCP) algorithm, whose aim is placing the FPGA clusters with high logic correlation near to one another [4]. The proposed placement algorithm places the design into the restricted contiguous regions. The results show that RCP can reduce the static power consumption by 19%.

Yazdanshenas et al. employed a static fine-grained power gating scheme in logic and routing resources to power off the unused SRAM cells [16]. They dedicate a configuration cell to control the power consumption of each switch box. In addition, they investigate the effect of dividing the SRAM cells of LUTs in power gating regions with various sizes on power saving. They achieve up to 75% reduction in the power consumption of logic blocks, however, they could only achieve less than 4% reduction in power consumption of switch boxes.

The SM multiplexers and their associated buffers and SRAM cells are equipped with power gating circuit in architecture proposed by Seifoori et al. [7]. Here, the authors extract the utilization pattern of routing resources in various routing architectures and investigate the effect of different power gating granularities. Based on their observations, they try to determine the most appropriate power gating granularity for each power gating architecture. Their experiments show that, by employing proper granularity, the routing power consumption can be decreased by 57%. However, the underlying routing architecture in their work is based on uniform wire lengths and simple switch patterns, which is not representative of modern FPGA routing architectures.

The composition of power gating region and their utilization pattern considerably affect the obtained power saving. Utilizing a heuristic approach to compose power gating region may result in a too few power gating opportunities and significant power overhead, and hence adverse the expected result. To alleviate the power overhead and increase the effectiveness of power gating approach in taking advantages of unused resources opportunities, we analyze the resource utilization patterns and then employ the data driven approaches to decide the optimum power gating region composition.

B. UTILIZING ML APPROACHES TO INCREASE THE EFFICIENCY OF FPGA DESIGNS

Previous research for optimizing the FPGA design by leveraging the ML approaches can be classified into two categories: (1) auto-tuning the FPGA design tool parameters and (2) increasing the quality of congestion estimation and routability prediction in FPGAs.

Optimizing and tuning the parameters of synthesis, map, and place-and-route design tools can significantly affect the target design in term of area, timing, and power metrics. However, due to enormous search space and long time needed to rebuild the design for different configurations, manual search for optimal parameters is impractical. Accordingly, the related works in the first category aim to optimize the configuration parameters using ML approaches. Mametjanov et al. leverage the linear regression and random forest to optimize the configuration parameters to tune the power consumption and performance of design [18]. Using Beysian classifier, Kapre et al. automate optimizing parameter selection through learning from a series of preliminary runs [19]. They reduce the FPGA timing closure by 70%. Xu et al. apply the multi-armed bandit technique to explore the large design space to autotune the tool-specific parameters, which control the complete FPGA design [20]. In addition, a parallelization scheme is used to parallelize exploring the complex design space. Yanghua et al. utilize the ML approach to predict the timing closure in FPGA design with focus on increasing the classification accuracy and decreasing the implementation iterations necessary in parameter selection [21]. Ustun et al. also autotune the design tools parameters utilizing the ML approaches [22]. In their proposed approach, instead of utilizing the extracted feature from the place-and-route iterations, the feature vector from the primitive steps of FPGA design flow is applied for learning in autotuning process, hence, the FPGA design closure is accelerated. Since power prediction is a challenging concern in hardware design, Lin et al. propose a learning-based power model to estimate the power consumption of FPGA applications [32]. To this end, a set of representative applications is used to construct training data in feature construction and their corresponding power collection, which is used to create a learning model to map the features to power estimation. The resource usage and timing reports estimated by high-level synthesis tools deviate significantly from the real results of the corresponding implementation on target FPGA. Due to the considerable importance of the resource usage and timing estimation in evaluating hardware design, Makrani et al. use machine learning approaches to achieve the throughput or throughput-to-area estimation of designs on target FPGA with high accuracy [33]. They first employ an automated hardware optimization tool to achieve the optimized metrics of the register-transfer level code, which is generated by a high-level synthesis tool, and then compose the training data set to employ in learning model to estimate the accurate design metrics. To increase the mapping efficiency of arithmetic operations on hardened blocks of FPGAs, Usten et al. propose to use graph neural network to automatically learn and extract the clustering pattern and operation mapping [34]. The training data is collected using the results of technology mapping of a set of microbenchmarks composed of arithmetic operations.

Congestion estimation and routability prediction in the placement stage can greatly improve the efficiency of placement and routing of the implemented design. By leveraging the simple regression technique, Qi et al. construct a congestion model for guiding the global router [23]. Applying the proposed model decreases the violation within design rule and routing runtime. Grewal et al. apply various clustering and regressoin techniques to model the relationship between different stages of FPGA design flow and the underlying behavior of the circuit to optimize the FPGA design [24]. The required training data is extracted from a broad range of 372 different benchmarks running on FPGAs with seven different academic configurations. To estimate the congestion, Pui et al. apply a linear regression model along with features based on wire length per area, pin count, and a feature related to the surrounding cells [25]. The results demonstrate that the accuracy of this model in predicting congestion is less than actual by 70%. The approach presented by Maarouf et al. increases the estimation accuracy through utilizing three new congestion related features with shorter runtime [35]. To estimate the routing congestion in high level synthesis, Zhao et al. propose a ML model to resolve congestion in source code through utilizing informative physical features [26].

III. BACKGROUND AND MOTIVATION

In this section, we give the basics of the FPGA architecture and *K*-means clustering and motivate our work.

A. FPGA ARCHITECTURE

Modern FPGAs consist of columns of configurable logic blocks and heterogeneous hardened units such as *Digital Signal Processing* (DSPs), external memory interfaces, processor cores, and transceivers [36], [37]. Fig. 1 illustrates one such heterogeneous FPGA architecture.

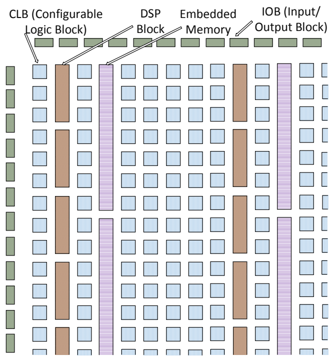

FIGURE 1. Common FPGA architecture. Besides logic blocks (in blue), FPGAs contain hard IP blocks (for example, DSPs), embedded memory blocks, external memory interfaces, transceivers, phased-locked loops, etc. Connectivity between all these elements is established using a configurable routing network, composed of horizontal and vertical interconnects and configurable switches.

The basic units of FPGAs are called *Configurable Logic Blocks* (CLBs), in Xilinx terminology, or *Logic Array Blocks* (LABs), in Intel terminology. LABs contain:

- *Logic Elements* (LEs), consisting of programmable *Look-Up Tables* (LUTs), flip-flops, and multiplexers providing additional connectivity.
- Intra-LAB connections between LE inputs, on one side, and the local feedback LE outputs and inter-LAB wires, on the other side.

The connections between FPGA building blocks are provided by the surrounding programmable routing fabric, composed of horizontal and vertical routing wires and switch matrices. Modern FPGA routing architectures use a mix of routing wires of various lengths, to balance the trade-off between area, delay, and flexibility of the routing network. For instance, the Intel Stratix-IV FPGA routing architecture has vertical wires spanning four and 12 rows of the FPGA array (V4 and V12, respectively), and horizontal wires spanning four and 20 columns (H4 and H20, respectively). The short wires are directly accessible by LAB inputs and outputs, while the long wires are accessible through both short and long wires, thanks to SM multiplexers.

The routing fabric of the Intel Stratix IV FPGA is illustrated in Fig. 2. As depicted in this figure, SMs have heterogeneous routing multiplexers: large (40:1) and small (12:1); these multiplexers drive the long and short wires, respectively (as illustrated in Fig. 3). Four different types of switch matrices can be identified in the architectural description of the Intel Stratix IV FPGA architecture, available as part of the VTR 8.0 suite [27]. They are listed in Table 1 and shown spatially distributed in Fig. 4.

B. MOTIVATION

Static power dissipation in FPGA routing network can be reduced using power gating, by clustering the routing resources into groups which, if unused, can be disconnected from the power supply. Clustering can be fine grained or coarse

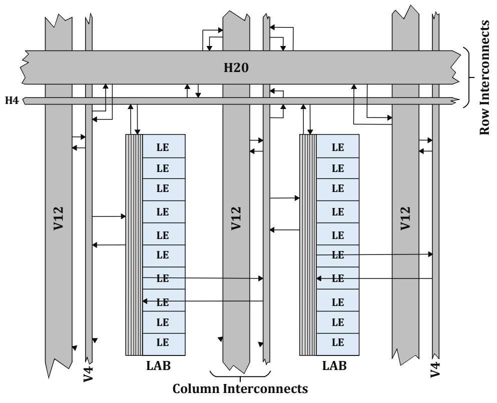

FIGURE 2. Connectivity between FPGA elements is achieved using FPGA routing resources: wires organized in horizontal and vertical routing channels and routing switch matrices, which enable wires to connect to each other. This figure illustrates a modern FPGA routing network based on the Intel Stratix IV FPGA architecture description available in VTR tool [27].

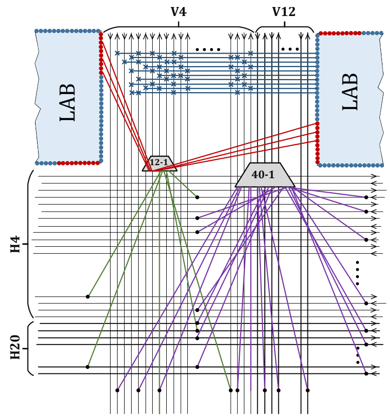

FIGURE 3. An illustration of a part of a switch matrix inside Stratix IV FPGA. It shows two multiplexer types (12:1 and 40:1) as well as two types of connections: (1) between column wires (V4, V12) and LAB inputs and (2) among LAB outputs, column interconnects, and row interconnects.

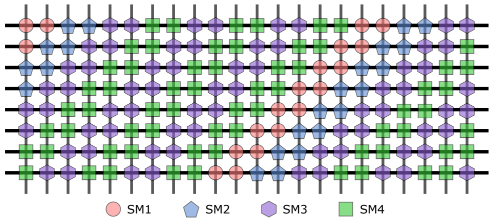

FIGURE 4. Spatial distribution of switch matrices in Stratix IV, shown on a small part of the FPGA. Horizontal and vertical lines represent routing channels, which are composed of unidirectional wires of nonuniform lengths (Fig. 2). Four different types of switch matrices can be identified (SM1, SM2, SM3, and SM4, listed in Table 1). They differ in the number of 12:1 and 40:1 multiplexers. A periodic pattern in switch matrix distribution can be observed, which is expected given that the wire lengths are all a multiple of four.

TABLE 1. Four routing switch matrix types extracted from the Stratix IV FPGA architecture description provided in VTR [38]. They are composed of 12:1 and 40:1 multiplexers, but the number of multiplexers per SM type differs, as the SM topologies are adapted for connecting the wire segments of nonuniform lengths.

Multiplexer	SM1	SM2	SM3	SM4
12:1	128	128	132	132
40:1	4	8	4	8

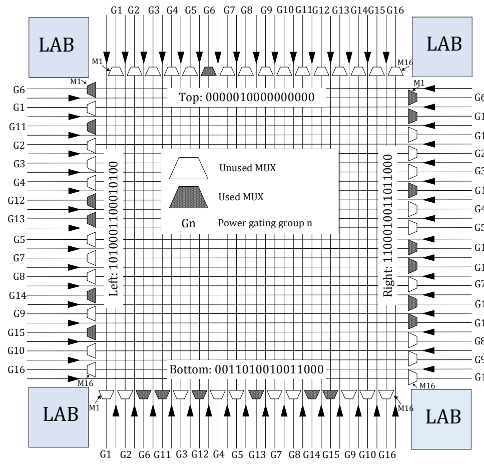
G1 G2 G3 G4 G5 G6 G7 G8 G9 G10 G11 G12 G13 G14 G15 G16
LAB
LAB
LAB
LAB
M1
M16
Top: 0000010000000000
Left: 1010001100010100
Right: 1100010011011000
Bottom: 0011010010011000
Unused MUX
Used MUX
Gn
Power gating group n
G6 G1 G11 G2 G3 G4 G12 G13 G5 G7 G8 G14 G9 G15 G10 G16
G6 G11 G1 G2 G3 G12 G4 G5 G13 G14 G7 G15 G16 G8 G9 G10
G1 G2 G6 G11 G3 G12 G4 G5 G13 G7 G8 G14 G15 G9 G10 G16

FIGURE 5. Distribution of used SM multiplexers and power gating groups (sample SM No. 1 in *usb-phy* circuit).

grained [2]–[7], [16], [17]; both approaches provide benefits at certain costs: fine granularity brings higher power gating opportunities at the cost of increased area and power overhead, while coarse granularity suffers from lower power gating opportunities, albeit at reduced area and power overhead. Additionally, for any granularity, clustering can be done in many different ways, which are not all equally effective.

Let us illustrate the challenge of clustering routing resources on an example of a simple FPGA routing architecture composed of wires of length one, arranged in 32-bit wide routing channels and connected to each other through switch matrices that contain a uniform number and type of multiplexers. Let us further consider *usb-phy* circuit, a sample benchmark from IWLS'05 benchmark suite [39] and use VTR suite to place and route this circuit. Fig. 5 shows one randomly chosen switch matrix (SM1), with routing multiplexers equally distributed on all fours sides and the multiplexers that are occupied by the benchmark circuit shaded in gray. Table 2 compares the utilization of routing multiplexers of SM1 and another randomly chosen switch matrix (SM2); in this table, zero stands for *unused* and one for *in use*. Clearly, the utilization of multiplexers varies among the switch matrices.

One clustering approach is to group the multiplexers driving the tracks with the same number (*id* in the channel) in all four sides of the switch matrix into a single power gating region; this strategy has previously been evaluated by Seifoori et al. [7]. The result is 16 clusters in Table 2: M1 corresponds to the multiplexers in the first cluster, which drive the wire segments of track number one in all four routing channels. Similarly, M2 is another cluster, which drives the wire segments of track number two and etc. This clustering approach can successfully turn off 25% of all clusters of SM1 (M5, M11, M15, and M16, in red); however, it can turn off only one power gating region in SM2 (M14, in red). This is equivalent to 36% and 11% of all **unused** multiplexers in SM1 and SM2, respectively. Table 3 presents another possible clustering scheme, in which we name each of the 16 power gating regions as Gi, $1 \leq i \leq 16$, and assign

TABLE 2. The utilization pattern of multiplexers in two randomly chosen switch matrices inside the FPGA region occupied by usb-phy benchmark. To place and route the benchmark, VTR [38] and a simplified FPGA architecture illustrated in Fig. 5 are used. In red, we highlight all power-gating clusters that can be powered off because they are fully unused.

SM SIDE	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
SM1-TOP	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SM1-RIGHT	1	1	0	0	0	1	0	0	1	1	0	1	1	0	0	0
SM1-BOTTOM	0	0	1	1	0	1	0	0	1	0	0	1	1	0	0	0
SM1-LEFT	1	0	1	0	0	0	1	1	0	0	0	1	0	1	0	0
SM2-TOP	0	0	1	1	0	1	1	0	0	1	1	0	0	0	0	1
SM2-RIGHT	1	1	1	0	1	0	1	0	0	1	0	1	1	0	0	1
SM2-BOTTOM	0	0	1	0	1	0	1	1	1	0	0	1	1	0	1	1
SM2-LEFT	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0

TABLE 3. Clustering labeled Gi in Fig. 5 is superior to clustering in Table 2, as more unused multiplexers are grouped in the same power gating regions. In red, we highlight all the clusters that can be powered off because they are fully unused.

SM SIDE	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16
SM1-TOP	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
SM1-RIGHT	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1
SM1-BOTTOM	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0
SM1-LEFT	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0
SM2-TOP	0	0	1	1	0	1	1	0	0	1	1	0	0	0	0	1
SM2-RIGHT	1	0	1	1	0	1	0	0	0	1	1	0	0	1	1	1
SM2-BOTTOM	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
SM2-LEFT	0	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0

to them the routing muxes as shown in Fig. 5. This new clustering can successfully switch off about 80% and 36% of all unused multiplexers in SM1 and SM2, respectively. Although the modified clustering provides considerable and acceptable results on this example, we have no guarantee that such equally acceptable results would be obtained if applied to all the remaining switch matrices. Therefore, to find a clustering approach that works well in general, it is crucial to examine a large set of applications. Given that finding an optimal clustering scheme is an NP-complete problem, we employ ML to find a near-optimal solution.

C. K-MEANS CLUSTERING

The growing need of knowledge discovery in the ever-increasing amount of data has lead to the advent of clustering algorithms; they play an important role in a wide variety of areas ranging from pattern classification [40] to knowledge discovery and data mining [41]. *K*-means algorithm is a widely acceptable unsupervised clustering algorithm, which has been extensively studied [42], [43].

K-means clustering partitions the data into a predetermined number of clusters so that a similarity metric (e.g., Euclidean distance) within clusters and between clusters is minimized and maximized, respectively [44], [45]. Generally, the similarity metric and the clustering objective function is chosen based on the application [46].

Let $\mathbf{V} = \{v_1, v_2, ..., v_N\}$ be a collection of N data objects and each $\mathbf{v}_i$ vector denoting the feature vector of $\mathbf{i}^{th}$ object. The purpose of *K*-means algorithm is clustering these data objects into K clusters $\mathbf{C} = \{C_1, C_2, ..., C_K\}$, where C_j corresponds to the j^{th} cluster centers. Initially, the cluster centers are selected arbitrarily. Then, the clustering algorithm proceeds in an iterative way, where each iteration is composed of the following steps:

1) Assign each data point x_i, $1 \leq i \leq N$, to the cluster C_j, $1 \leq j \leq K$, whose center has the least squared Euclidean distance to x_i [47]–[49].
2) Compute the cluster center as means of all the members.

The iterations continue until the convergence is reached (i.e., the cluster centers do not change in consecutive algorithm iterations).

K-means clustering quality strongly depends on three user-specified parameters: a) the total number of clusters (i.e., k), b) the homogeneity metric, and c) the cluster centers initialization. Decision making on the number of clusters is often carried out in an *ad hoc* manner, depending on the problem definition, prior knowledge, and experiments. Alternatively, one can run *K*-means with variable number of clusters and then select the number of clusters with the most promising results. The most dominant homogeneity metric in various versions of *K*-means clustering algorithm is the distance measure; however, other homogeneity metrics, in particular if based on the target application, can replace the distance metric.

Since the *K*-means algorithm converges to the local minima, different initial configurations can result in unequally effective clustering. A large number of techniques have been proposed to enhance the robustness of *K*-means clustering algorithm against the sensitivity toward the initialization. To overcome the local minima problem, one can run *K*-means for various cluster initialization and select the one with the least squared error. Employing this approach when dealing with large data sets incurs a high computation cost. Other approach is, for instance, global *K*-means clustering: in each iteration, deterministically find the optimal candidate for each cluster center among the data objects through comparing the clustering results of various *K*-means runs versus assigning each data object as the center of the desired cluster [50]. This algorithm is computationally demanding as well, because it runs the *K*-means clustering several times for finding each cluster center.

Yet another approach is to select the center of the first cluster randomly and the centers of other clusters based on their distance to the previously selected centers; this approach is called k-means++ [51]. The time complexity of this algorithm is acceptable, in comparison with other variations. Moreover, Arthure et al. [51] guarantee the existence of an approximation of optimum (i.e., $logk$) for k-means++. More precisely, let us consider the potential function for an arbitrary clustering C, defined as the total squared distance between each data point and its closest center, as follows:

$$\phi = \sum_{x \epsilon X} min_{c \epsilon C} \|x - c\|^2 \tag{1}$$

and let C_{OPT} denote the optimal clustering for a set of data points. Accordingly, If k-means++ constructs the cluster C, one can prove that the corresponding potential function satisfies the following:

$$E[\phi] = 8(\ln k + 2)\phi_{OPT} \tag{2}$$

Hence, in this work, we employ k-means++ approach to initialize the cluster centers.

IV. OUR CLUSTERING ALGORITHMS

As discussed in Section III-B, obtaining an optimal or near-optimal multiplexer clustering in one switch matrix does not guarantee equally acceptable results in other switch matrices. In addition, looking for the optimal multiplexer clustering through implementing all possible clustering solutions and their comparison is impractical, due to the prohibitively large solution space. The above issues are common in data-analysis problems, frequently solved using machine learning techniques [52]–[56]. Hence, instead of employing empirical approaches to cluster the switch matrice multiplexers (as done previously), we opt for using an ML approach (e.g., K-means). Additionally, we derive here a new ML-based algorithm for efficiently addressing the power-gating challenge.

The first step of our clustering approach is extracting the training data from the input set of L learning benchmarks, i.e., building a set of *feature vectors* as follows:

1) Place and route L learning benchmarks on the target FPGA architecture.
2) Find all the used switch matrices[2] inside the FPGA region occupied by the learning benchmark. If the FPGA routing resources employ more than one type of switch matrices, e.g., some SMs differ in the number or size of multiplexers as is the case in Intel Stratix-IV architecture, then all the unique SM types need to be identified and treated as independent clustering problems.
3) Build feature vectors v_i for each multiplexer M_i of a switch matrix as the following row vectors:

$$v_i = \begin{bmatrix} v_{i_1} & v_{i_2} & \dots & v_{i_L} \end{bmatrix}. \tag{3}$$

Here, v_{i_n} is a row vector that corresponds to the utilization of multiplexer M_i (1 if in use, 0 otherwise) in all switch matrices of n^{th} benchmark. The element in the j^{th} column of vector v_{i_n} therefore corresponds to the utilization of the multiplexer M_i in the j^{th} instance of the switch matrix in benchmark B_n: 1 if M_i is in use by the benchmark B_n, 0 otherwise. Given that the number of used switch matrices in learning benchmarks is not the same across all benchmarks, the length of vectors v_{i_n} is not constant either.
4) Add vectors v_i to the training data set V.

[2]The switch matrices with at least one used multiplexer.

In the remainder of this section, we first present a clustering algorithm that uses K-means algorithm to build power-gating clusters (Section IV-A). Then, we introduce a novel metric called *utilization similarity* and derive new clustering algorithms (Section IV-B). Finally, we modify the FPGA routing algorithm [57] to account for the presence of power-gating clusters and use them efficiently (Section IV-C).

A. CLUSTERING USING K-MEANS ALGORITHM (KM)

Our first power-gating approach is a straightforward implementation of K-means clustering, following the steps of Algorithm 1. Given the training data set V and the desired number of power-gating regions K, the algorithm starts by initializing *cluster centers*. Then, it empties all clusters, assigns the multiplexers to the closest clusters (smallest Euclidean distance to the cluster center), and recomputes the cluster centers. These steps are repeated until there are no changes in the cluster composition (i.e., the multiplexers assigned to the clusters do not change from one iteration to another) or until the maximum number of iterations is reached.

To initialize the centers of the cluster, we employ k-means++, one of the most commonly used initialization approaches [58] (Section III-C). `InitCenters` function, shown in Algorithm 2, starts by initializing the center of the first cluster μ_1 with a randomly chosen element from the set V. Then, for every v_i in V, it calls `ComputeDistance` to find the Euclidean distance $d_{i,j}$ between v_i and an already initialized cluster center μ_j:

$$d_{i,j} = ||v_i - \mu_j||^2 \tag{4}$$

Finally, it computes the probability that v_i may become the new cluster center as $\frac{d_{i,j}^2}{\sum_{i=1..N} d_{i,j}^2}$, where N is the cardinality of set V. Vector v_i that maximizes this probability is then chosen as the initial center of the next cluster.

At the end of each iteration of the K-means clustering algorithm, we call `UpdateCenter` function to find the mean of all cluster members and update the cluster center accordingly.

B. CLUSTERING USING UTILIZATION SIMILARITY METRIC (SIM)

A power gating region is a collection of switch matrix multiplexers controlled by a common power-gating circuit. As shown earlier, multiplexers can be described using a feature vector in Eq. (3), where each dimension reflects the utilization of the corresponding multiplexer in one of

Algorithm 1: Power gating using K-means clustering.

```
Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: $K$
Output: Clusters $C = \{C_1, C_2, ..., C_K\}$
Variables: Cluster centers: $\mu_1, \mu_2, ..., \mu_K$
InitCenters($V$, $K$)
while Cluster elements change do
    foreach $C_i \in C$ do
        $C_i \leftarrow \emptyset$
    foreach $v_i \in V$ do
        $k \leftarrow$ TheClosestCluster($v_i$, $C$)
        $C_k \leftarrow C_k \cup v_i$
        $\mu_k \leftarrow$ UpdateCenter($\mu_k$, $v_i$)
```

Algorithm 2: Function `InitCenters` for computing initial cluster centers.

```
Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: $K$
Output: Cluster centers: $\mu_1, \mu_2, ..., \mu_K$
Variables: Distance between $v_i$ and $\mu_j$: $d_{i,j}$; sum of
           distances squared: $dssq$; next cluster ID: $k$;
           probabilistic distance metric: $p$
$\mu_1 \leftarrow$ randomly chosen from $V$
$k \leftarrow 2$
while $k < N$ do
    $dssq \leftarrow 0$
    foreach $v_i \in V$ do
        foreach $\mu_j$, $1 \le j < k$ do
            $d_{i,j} \leftarrow$ ComputeDistance($v_i$, $\mu_j$)
            $dssq \leftarrow dssq + d_{i,j}^2$
    $p \leftarrow 0$
    foreach $v_i \in V$ do
        $p_{i,j} \leftarrow \frac{d_{i,j}^2}{dssq}$
        if $p_{i,j} > p$ then
            $p \leftarrow p_{i,j}$
            $\mu_k \leftarrow v_i$
    $k \leftarrow k + 1$
```

the switch matrices (1, if used; 0, otherwise). Consequently, two feature vectors v_i and v_j are identical only if their two corresponding switch matrix multiplexers have exactly the same utilization pattern across all learning benchmarks; there is no doubt that these vectors should be assigned to the same power gating region. However, in most practical cases, the feature vectors differ significantly. To address this, we introduce *utilisation similarity metric* as follows:

$$s_{i,j} = \sum_{m=1..|v_i|} \begin{cases} 1, & \text{if } v_i[m] = v_j[m] \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

This metric, computed on a pair of feature vectors v_i and v_j, is equal to the number of dimensions m in which these two vectors are the same: the higher it is, the more beneficial it is to keep the two corresponding multiplexers inside the same cluster.

The notion of the cluster center, presented in Section IV-A, does not fit well the utilisation similarity metric in (5). An alternative could be to use a cluster *mode*, common in categorical data clustering. A cluster mode is a vector whose elements correspond to the most frequent value of that dimension across all feature vectors of the cluster members. However, this is not suitable either, because the majority of the elements of the feature vectors in (3) are zeros, due to the large number of unused multiplexers in the FPGA switch matrices; consequently, the resulting cluster modes would lean towards zero. Let us introduce the notion of a cluster *pattern* ρ, as a measure of similarity between all cluster members. Naturally, the cluster pattern of a single-member cluster is equal to that member. Then, when assigning each new member to the cluster, the cluster pattern ρ is updated based on the feature vector v of the newly added element as follows:

$$\rho[m] = \begin{cases} 1, & \text{if } \rho[m] = v[m] \text{ and } v[m] = 1 \\ 0, & \text{if } \rho[m] = v[m] \text{ and } v[m] = 0 \\ \mathbf{X}, & \text{if } \rho[m] \neq v[m], \end{cases} \tag{6}$$

such that $1 \le m \le |v|$. Here, $\mathbf{X}$ indicates a value other than zero and one[3]. Therefore, we can express the power gating efficiency ε of cluster C_i as the product of the cluster size and the number of pattern elements different from $\mathbf{X}$.

$$\varepsilon(C_i) = |C_i| \cdot \Big(|\rho_i| - \texttt{CountX}(\rho_i)\Big). \tag{7}$$

The higher the value of $\varepsilon(C_i)$, the higher the similarity between the cluster elements. The power gating efficiency of all K clusters then becomes:

$$\varepsilon(C) = \sum_{i=1..K} |C_i| \cdot \Big(|\rho_i| - \texttt{CountX}(\rho_i)\Big). \tag{8}$$

Since the expression in (7) is nonnegative, the higher the value of $\varepsilon(C)$, the better the clustering.

In the following subsections, we present four novel approaches towards efficient clustering and power gating of the FPGA routing multiplexers.

1) SiM clustering

Our first clustering algorithm, which we name *SiM* (i.e., similarity clustering), is summarized in Algorithm 3. This clustering aims to maximize the power gating efficiency by using the cluster patterns instead of the cluster centers and the similarity metric instead of the Euclidian distance. The algorithm starts by calling the function `InitPatterns` to initialize the cluster patterns. This function is similar to `InitCenters`; however, it replaces the call to `ComputeDistance` with the computation of the similarity metric in Eq. (5).

Then, for every v_i in V, we search for the cluster yielding the highest similarity metric between v_i and the cluster pattern, to insert v_i in it. Finally, we update the cluster pattern following Eq. (6).

[3]In our implementation, without loss of generality, $\mathbf{X}$ is set to -1.

Algorithm 3: Our proposed *SiM* clustering, which uses similarity metric defined in Eq. (5) and the concept of cluster pattern from Eq. (6).

```
Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: $K$
Output: Clusters $C = \{C_1, C_2, ..., C_K\}$
Variables: Cluster patterns: $\rho_1, \rho_2, ..., \rho_K$
InitPatterns($V$, $K$)
foreach $v_i \in V$ do
    $k \leftarrow$ TheMostSimilarCluster($v_i$, $C$)
    $C_k \leftarrow C_k \cup v_i$
    $\rho_k \leftarrow$ UpdateClusterPattern($\rho_k$, $v_i$)
```

Algorithm 4: Our proposed *SiM-PR* clustering. Unlike *SiM*, this algorithm runs multiple iterations, at the end of every iteration, replaces the cluster pattern with a randomly chosen cluster member. As a consequence, at the end of every iteration, the patterns elements are reduced back to two values only: 0 and 1.

```
Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: $K$
Output: Clusters $C = \{C_1, C_2, ..., C_K\}$
Variables: Cluster patterns: $\rho_1, \rho_2, ..., \rho_K$
InitPatterns($V$, $K$)
while Cluster elements change do
    foreach $v_i \in V$ do
        $k \leftarrow$ TheMostSimilarCluster($v_i$, $C$)
        $C_k \leftarrow C_k \cup v_i$
        $\rho_k \leftarrow$ UpdateClusterPattern($\rho_k$, $v_i$)
    foreach $C_i \in C$ do
        $\rho_i$ = RandomElementFromCluster($C_i$)
```

It should be noted that starting new clustering iteration with existing cluster pattern is rather unlikely to result in an efficient clustering solution, because CountX(ρ_i) would either remain the same or increase as iteration advance. To iteratively repeat Algorithm 3, similar to *K*-means clustering, we need a strategy for updating the cluster patters between subsequent iterations. This is addressed in our next clustering approach.

2) SiM-PR clustering

To render the *SiM* algorithm iterative, we choose to replace the cluster pattern with a randomly selected member of the cluster, at the end of each iteration of the algorithm. We call this new algorithm *SiM* with pattern reduction (*SiM-PR*), because the cluster pattern is *reduced* to another cluster member. The *SiM-PR* approach is detailed in Algorithm 4.

3) SiM-IPR clustering

SiM and *SiM-PR* result in two extreme solutions. The former completes only one iteration, while the latter allows for more iterations but, at the end of every iteration, it reduces cluster patterns to randomly chosen elements from the corresponding clusters. The randomness in *SiM-PR* can help finding a more efficient clustering solution compared to *SiM*, but it can also cause the algorithm to remain in a locally-optimal solution. Therefore, we propose an alternative in which the cluster patterns are reduced incrementally, depending on the efficiency metric defined in Eq. (7). We name this algorithm *SiM* with Incremental Pattern Reduction (*SiM-IPR*) and show its implementation in Algorithm 5. Unlike *SiM-PR*, this algorithm applies pattern reduction only on R ($R < K$) clusters that have the lowest power gating efficiency metric, which we compute using Eq. (7). The parameter K stands for the total number of clusters, while the parameter R (rate) is a variable, equal to the number of clusters whose patterns are to be reduced in the current clustering iterations. In our implementation, we choose to set the initial value of R to $K/2$ and, in every subsequent iteration, to reduce R by half.

Algorithm 5: Our proposed *SiM-IPR* clustering. Unlike *SiM-PR*, this algorithm applies pattern reduction only on R clusters that are the most inefficient, according to Eq. (7). The parameter R stands for rate. Its initial value is $K/2$. In every subsequent iteration, it is reduced by half.

```
Input: Data set: $V = \{v_i\}, i = 1..N$
Input: Total number of clusters: $K$
Output: Clusters $C = \{C_1, C_2, ..., C_K\}$
Variables: Cluster efficiencies: $E = \{E_1, E_2, ..., E_K\}$;
           set of clusters whose patterns are to be
           reduced: $C_R$; cluster reduction rate: $R$; cluster
           patterns: $\rho_1, \rho_2, ..., \rho_K$.
InitPatterns($V$, $K$)
$R \leftarrow K/2$
while Cluster elements change do
    foreach $v_i \in V$ do
        $k \leftarrow$ TheMostSimilarCluster($v_i$, $C$)
        $C_k \leftarrow C_k \cup v_i$
        $\rho_k \leftarrow$ UpdateClusterPattern($\rho_k$, $v_i$)
    foreach $C_i \in C$ do
        $E_i \leftarrow$ Efficiency($C_i$)
    $C_R = \emptyset$
    for $1 \leq r \leq R$ do
        $k$ = LeastEfficientIndex($E$)
        $E \leftarrow E \setminus E_k$
        $C_R = C_R \cup C_k$
    foreach $C_i \in C_R$ do
        $\rho_i$ = RandomElementFromCluster($C_i$)
    $R \leftarrow R/2$
```

4) SiM-IPR-MP clustering

The clustering algorithms proposed in Sections IV-B1, IV-B2, and IV-B3 focus on grouping multiplexers with the highest utilization similarity, without taking into account the effects such a decision can have on the actual power consumption of the entire switch matrix. The consequences we refer to here are two-fold: first, adding a mux to a cluster can affect the cluster pattern and, therefore, its power-saving opportunities. Second, adding a mux to a cluster increases the cluster size and the overall power consumption of the cluster; the latter corresponds to the sum of the power consumption of the cluster members and the power consumption of the power-gating circuit itself.

Before proceeding, let us recall the meaning of dimensions equal to zero in the cluster patterns (Eq. (6)): they correspond to the muxes that are *unused* by all the cluster elements and, consequently, can be powered off. For a cluster pattern of length N and having N_0 dimensions equal to zero, we can thus say that the *probability* of this particular cluster to be powered off equals the probability that the circuit to be programmed onto the FPGA uses one of the muxes corresponding to those N_0 dimensions:

$$P_{\mathrm{OFF}}(C_i) = \frac{N_0}{N}. \tag{9}$$

The probability of a cluster being powered on is equal to:

$$P_{\mathrm{ON}}(C_i) = 1 - P_{\mathrm{OFF}}(C_i) = \frac{N - N_0}{N}. \tag{10}$$

The static power consumption of a power-gating cluster C_i can be approximated as the weighted sum:

$$W(C_i) = P_{\mathrm{OFF}}(C_i) \cdot W_{\mathrm{OFF}}(C_i) + P_{\mathrm{ON}}(C_i) \cdot W_{\mathrm{ON}}(C_i) \tag{11}$$

where $W_{\mathrm{OFF}}(C_i)$ and $W_{\mathrm{ON}}(C_i)$ represent the static power consumption of the power-gating cluster when it is powered on and powered off, respectively. $W_{\mathrm{ON}}(C_i)$ corresponds to the sum of the power consumption of the power-gating circuitry, $W_{PG_{ON}}$, and the power consumption of the multiplexers in the cluster, when the cluster is powered on. Similarly, $W_{\mathrm{OFF}}(C_i)$ is equal to the power consumption of the power-gating circuit alone, $W_{PG_{OFF}}$, when the cluster is powered off, which brings us to the next expression:

$$W(C_i) = P_{\mathrm{OFF}}(C_i) \cdot W_{PG_{OFF}}(C_i) + \\ + \Big(1 - P_{\mathrm{OFF}}(C_i)\Big)\Big(|C_i| \cdot W_{\mathrm{MUX}} + W_{PG_{ON}}(C_i)\Big). \tag{12}$$

Here, W_{MUX} is the power consumption of each multiplexer in cluster C_i. Consequently, the power consumption of the entire routing switch matrix with K power gating clusters can be approximated as:

$$W(SM) = \sum_{i=1}^{K} W(C_i) \tag{13}$$

Let us now turn to the example shown in in Fig. 6, where we need to decide whether to assign multiplexer M1 (highlighted in red) to cluster G of size three or to cluster F of size seven. Supposing that the utilization similarity between M1 and cluster G is higher than between M1 and cluster F, our previously proposed clustering algorithms would naturally assign M1 to cluster G. However, let us further assume that P_{ON} of cluster G and of cluster F, after assigning M1 to them, are 20% and 50%, respectively. If we run a HSPICE simulation to estimate $W_{PG_{\mathrm{ON}}}$, $W_{PG_{\mathrm{OFF}}}$, and W_{MUX}, to

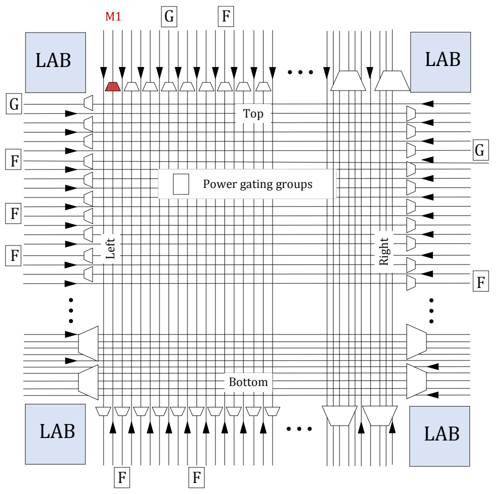

FIGURE 6. An illustration of an FPGA routing switch matrix, with G and F as example power-gating clusters. M1, highlighted in red, is a multiplexer that needs to be assigned to one of them.

compute the value of expression in Eq. (12) for clusters G and F, we find that a better decision would be to assign M1 to F, because it would result in higher power saving (1.75 nW power saved for M1 in F versus 1.39 nW power saved for M1 in G). Therefore, we find that relying on utilization similarity only is not the optimal criteria: we need to take into account the impact that the clustering decision has on the overall power consumption—something that we ignored in the previously proposed clustering algorithms.

In this section, we propose a clustering algorithm named *SiM with Incremental Pattern Reduction to Minimize Power consumption* (*SiM-IPR-MP*), in which we assign multiplexers to clusters with the most utilization similarity as well as the least power consumption increase, i.e., the least difference in the static power consumption after and before assigning the multiplexer to a cluster. To get there, we will first discuss how we estimate the power consumption of the power-gating circuitry, and then present the derivation of the new clustering metric.

(a) Power consumption of power-gating circuitry: For the purpose of estimating the static power consumption of the power-gating circuit, we use HSPICE and run a set of circuit-level simulations, employing *Predictive Technology Model* (PTM) [59] and the exact FPGA transistor sizing extracted through COFFE [29]. Fig. 7 shows our proposed power gating circuit-design approach. As can be seen, the sleep transistors are inserted between the main power supply and the virtual power supply. The size of sleep transistors depends on the number of multiplexers in each cluster, and is chosen so that it does not affect their delay. We vary the number of multiplexers in the cluster $|C_i|$ and record the power consumption of the entire cluster, including the power-gating circuit, when it is powered on or powered off. The obtained results for the cluster in powered-on state are plotted in Fig. 8, in blue. Given the strong linear dependency between the power consumption and

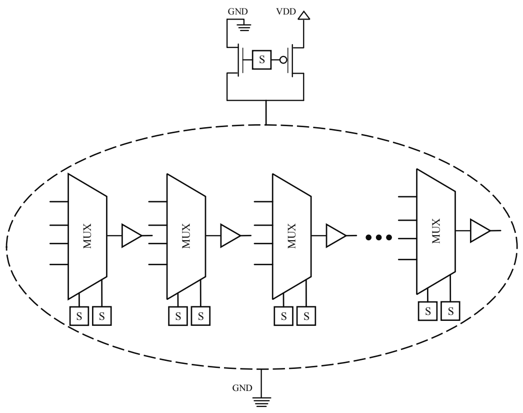

FIGURE 7. The structure of the proposed power gating scheme. The dashed line depicts one power gating region. The number of multiplexers inside the power gating regions can vary based on the clustering algorithm. The power state of each power gating region is controlled by a SRAM cell, which can turn on the PMOS or NMOS sleep transistor.

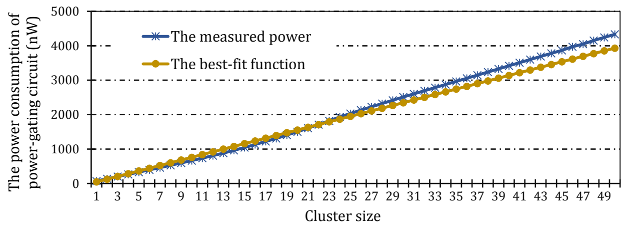

FIGURE 8. In blue, the power consumption of the power-gating circuit when the cluster is powered off, in the function of cluster size. We can notice that increasing the cluster size results in higher power consumption of the power-gating circuit. In yellow, the best-fit linear approximation.

the cluster size, we use a data analysis software[4] to find a best-fit linear function (shown in yellow in the same figure), which we will later use to derive the clustering metric.

The following best-fit linear approximation function is obtained:

$$F(|C_i|) = a \cdot |C_i| + b, \tag{14}$$

where a is 79.3 nW and b is -33.4 nW and the approximation error is below 1.14%. We then repeat the simulation in HSPICE, this time with the cluster powered off, and find that the power consumption of the power-gating circuit is approximately doubled. Therefore, for simplicity, instead of computing the power consumption of the power-gating circuit when the cluster is powered off, we use the same linear best-fit function as in Eq. (14) with the coefficients multiplied by a factor of two.

Following the expressions (13) and (14), we can write:

$$W(SM) = \sum_{i=1}^{K} \Big(P_{\text{OFF}}(C_i) \cdot (2a \cdot |C_i| + 2b) + \\ +(1 - P_{\text{OFF}}(C_i))(W_{\text{MUX}} \cdot |C_i| + a \cdot |C_i| + b) \Big) \tag{15}$$

(b) Derivation of new clustering metric: Adding a multiplexer to a cluster affects the overall power consumption. This change in power consumption can be found by computing the following expression:

$$\Delta W(C_i) = W_{\text{IN}}(C_i) - W_{\text{OUT}}(C_i). \tag{16}$$

Here, $W_{\text{IN}}(C_i)$ is the power consumption of the cluster C_i after the multiplexer is added to the cluster. Similarly, $W_{\text{OUT}}(C_i)$ is the power consumption of the cluster before the multiplexer is added. These two values are related to the number of dimensions equal to zero in the cluster pattern (N_0), the pattern length N, and the impact of adding the multiplexer to the cluster on the number of dimensions equal to zero (ΔN_0):

$$P_{\text{OFF,IN}} = \frac{N_0}{N}, \\ P_{\text{OFF,OUT}} = \frac{N_0 - \Delta N_0}{N}. \tag{17}$$

Following the expressions (14) and (12), equation (16) becomes:

$$\Delta W(C_i) = P_{\text{OFF,IN}}[2a(|C_i|+1) + 2b] + \\ +(1 - P_{\text{OFF,IN}})[W_{\text{MUX}}(|C_i|+1) + a(|C_i|+1) + b] - \\ -P_{\text{OFF,OUT}}(2a\,|C_i| + 2b) - \\ -(1 - P_{\text{OFF,OUT}})[W_{\text{MUX}}\,|C_i| + a\,|C_i| + b]. \tag{18}$$

After applying the expression in (17) to the previous equation, we obtain:

$$\Delta W(C_i) = \frac{1}{N}\Big\{\Delta N_0[(W_{\text{MUX}} - a)(|C_i|+1) - b] - \\ -N_0(a - W_{\text{MUX}})\Big\} + W_{\text{MUX}} + a. \tag{19}$$

Finally, knowing that W_{MUX}, a, and N are constant, minimizing ΔW becomes equivalent to minimizing the following simplified objective function:

$$\mathcal{W} = \Delta N_0[(W_{\text{MUX}} - a)(|C_i|+1) - b] - \\ -N_0(a - W_{\text{MUX}}) \tag{20}$$

(c) Distinguishing between small and big multiplexers: In all previously derived expressions, we use a single variable to denote power consumption of a routing multiplexer, W_{MUX}. However, not all multiplexers in routing switch matrices are necessarily of the same size. As discussed in Section III-A, in the architectural model of Stratix-IV FPGA, we uncover that the switch matrices are composed of two types of multiplexers, different in size.

Therefore, their power consumption differs: it is higher for the large multiplexers. Hence, particularly high priority should be given to efficiently clustering the large multiplexers. In adding a new multiplexer to a cluster, two scenarios should thus be distinguished:

1) The new multiplexer is small, in which case the power consumption increase caused by assigning the multiplexer to the cluster can be computed from (18).

2) The new multiplexer is large, capable of providing $\mathcal{M}$-times more power saving if clustered well, i.e., if it can be powered off. Here, $\mathcal{M}$ is the ratio of the power consumption

[4]CurveExpert Professional

of a large multiplexer over the power consumption of a small one, which we compute in HSPICE simulation. Hence, adding a large multiplexer becomes equivalent to adding $\mathcal{M}$ small multiplexers to the cluster. Accordingly, the equation (18) becomes:

$$\Delta W = \frac{1}{N}\Big\{\Delta N_0[(W_{\text{MUX}} - a)(|C_i| + \mathcal{M}) - b] + \\ + \mathcal{M}\cdot N_0(a - W_{\text{MUX}})\Big\} + \mathcal{M}\cdot W_{\text{MUX}} + \mathcal{M}\cdot a \quad (21)$$

Once again, given that W_{MUX}, N, a, and b are constants, minimizing the power consumption increase due to the inclusion of a large multiplexer in the cluster corresponds to minimizing the following metric:

$$\mathcal{W}_{\text{LARGE}} = \Delta N_0[(W_{\text{MUX}} - a)(|C_i| + \mathcal{M}) - b] - \\ - \mathcal{M}\cdot N_0(a - W_{\text{MUX}}) \quad (22)$$

The algorithm implementing the described clustering is, in its essence, very similar to Algorithm 5. The main difference is in the computation of the similarity metric and in the fact that the metric differs in the function of the multiplexer size.

C. POWER-GATING AWARE ROUTING

Once the power-gating regions are implemented in the FPGA fabric, it seems natural to try to make the best use of them; that is precisely our next step.

Let us take all the benchmarks in Table 4—we will explain them in more detail in the upcoming section—and apply *SiM-IPR-MP* clustering to find the power-gating regions. Then, let us take the resulting FPGA switch-matrix architecture, and use it to place and route the benchmarks. After counting the used (occupied) versus unused routing multiplexers, we arrive to the plot shown in Fig. 9. Interestingly, the vast majority of power-gating regions (79.3%) has low utilisation rate: less than 25% of their muxes are used. Additionally, we find that there are only around 5% of all the power-gating regions where at least half of the multiplexers are used.

If there would be a possibility to guide the FPGA router away from the fully unused power-gating regions and towards those that are already in use (i.e., cannot be powered off), higher power saving could be achieved. This idea is illustrated in the example in Fig. 10, showing two power-gating regions. The first region is composed of five multiplexers shaded in dark grey; 80% of them are already occupied (dashed red lines correspond to the already routed connections). The second region is composed of six multiplexers shaded in light grey. If the FPGA router could avoid the second region by replacing the tentative connection marked as (1) with the connection marked as (2), then the second power-gating group would remain fully unused and, consequently, it could be powered off. To make the best use of the available power-gating regions, we need to enhance the FPGA routing algorithm: make it aware of the existence of the power-gating regions and have it use them efficiently.

(a) FPGA routing problem and algorithm: The primary data structure representing FPGA routing resources is the directed

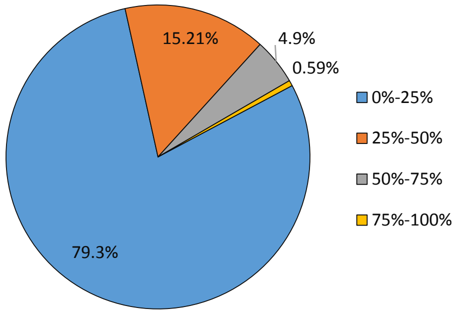

FIGURE 9. The rate of resource utilization in power-gating regions: In the vast majority of power gating groups (in blue) less than a quarter of the resources is used. In addition, the rate of highly-used power gating groups is lower than 1% (in yellow).

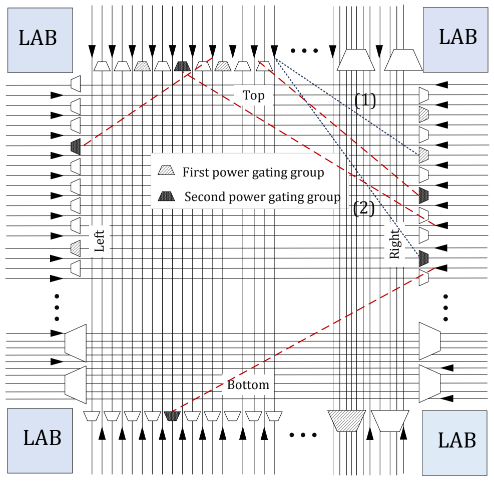

FIGURE 10. The role of routing modification in increasing the power gating opportunities. Here, by modifying the routing of nets (replacing the connection (1) with the connection (2)), we can extract some unused groups to act as power gating opportunities (the first power gating group).

Routing Resource Graph (RRG) $G = (V, E)$, where V is the set of vertices and E is the set of edges. Each vertex $v \in V$ represents wires and pins that are internal to FPGA. Each edge $e_{ij} \in E$ represents a programmable connection point between a pin and a wire segment, or a programmable routing switch between two wire segments. Each signal i to route through G forms a net $N_i = (s_i, \{t_{i,1}, t_{i,2}, ..., t_{i,m}\})$, where s_i is the net source vertex and $\{t_{i,1}, t_{i,2}, ..., t_{i,m}\}$ are the sinks. The solution to the routing problem of the net N_i is a set of paths from the source s_i to all the net sinks; these paths form a directed *routing tree* $RT(N_i) \subset G$. Routing is successful if the routing trees of different nets are disjoint in G.

In 1995, Mc Murchie and Ebeling presented PathFinder, an iterative algorithm that achieves a good compromise between two conflicting goals: eliminating congestion and minimizing critical path delay [57]. Its distinctive feature is the existence of a cost of using a given vertex v in a route. This cost depends on the delay of the vertex and its congestion history. While signals compete for vertex v, they *negotiate* and the cost evolves as the algorithm runs. This algorithm, also called *negotiation-based router*, is used in the modern commercial and research FPGA design tools, VTR included. PathFinder implementation is a triple-nested loop:

- The outer loop (all-net router), invokes the middle loop (signal router), for all signals i to be routed. If there is no congestion, i.e., no routing trees share routing resources, PathFinder terminates. Additionally, if a user-defined number of iterations is exceeded, PathFinder terminates. Otherwise, before proceeding to the next iteration, all-net router updates the second-order (also called historical or accumulated congestion) costs $h(v)$ of all congested nodes.
- Each signal router iteration starts by riping up the existing routing tree $RT(N_i)$ of a net N_i. As a consequence, the occupancy of nodes in $RT(N_i)$ is decreased and their first-order (also called present congestion) costs $p(v)$ are updated accordingly. Then, it invokes the inner loop (maze expansion) to re-route the net N_i. Once new routing tree is created, the present congestion costs of all congested nodes in $RT(N_i)$ are updated.
- Maze expansion traverses the RRG, starting from the source node of a net. It initializes the routing tree $RT(N_i)$ with the source node. Then, it expands the source node, i.e., uncovers all its neighbors and stores them in a *Priority Queue* (PQ) sorted by their costs. These costs are a function of a) the node base cost $b(v)$ (equal to the delay of the node, $delay(v)$), b) its present cost $p(v)$, c) its accumulated cost $h(v)$, and d) the net criticality:

$$\begin{aligned} Cost(v) = Crit(i,j) \cdot delay(v) + \\ +\Big(1 - Crit(i,j)\Big) b(v)h(v)p(v). \end{aligned} \tag{23}$$

The first term in (23) is a delay-sensitive term, while the second is congestion-based. The criticality of a net is equal to:

$$Crit(i,j) = 1 - \frac{slack(i,j)}{D_{max}}, \tag{24}$$

where D_{max} denotes the delay of the critical path of the circuit and $slack(i,j)$ is the slack of source and sink j connection of net i. In each subsequent maze expansion iteration, the lowest-cost vertex v_{min} is extracted from PQ. If v_{min} is a sink of the net N_i, a path is constructed by invoking a backtrace procedure and added to $RT(N_i)$. Otherwise, v_{min} is expanded and all its neighboring nodes which have not been previously visited are inserted in the PQ. Maze expansion continues until paths to all sinks are found and the routing tree is complete.

(b) Power-gating-aware FPGA routing: To make the routing algorithm aware of the existence of power-gating clusters, we introduce two enhancements. The first modification concerns the routing resource graph: for every vertex v that corresponds to a switch-matrix routing multiplexer, we introduce an additional cost named *power-gating cost*, or $\mathcal{PG}(v)$.

If the power-gating cluster to which the multiplexer v belongs is already in use (i.e., at least one of its members is occupied), then the cost $\mathcal{PG}(v)$ is cleared, to prevent it from having an impact on the FPGA routing algorithm. If, however, the power-gating cluster is not used (i.e., entirely unoccupied), then we choose to scale the cost with the cluster size and the routing iteration i as follows:

$$\mathcal{PG}(v) = \begin{cases} 0, & \text{if cluster is in use,} \\ b(v) \cdot |C(v)| \cdot i\,, & \text{otherwise.} \end{cases} \tag{25}$$

By scaling the cost with the cluster size and the iteration count, we strongly encourage the router to avoid routing through the unused power-gating regions. Scaling with the iteration count is common; it is applied by PathFinder when updating the first-order $p(v)$ and the second-order $h(v)$ costs.

The second enhancement we introduce concerns the computation of the total cost of a routing multiplexer, shown in (23). To avoid the new cost function directly affecting the circuit critical path delay, we choose to augment the congestion-based part of the cost, by including $\mathcal{PG}(v)$:

$$\begin{aligned} Cost(v) = Crit(i,j) \cdot delay(v) + \\ +\Big(1 - Crit(i,j)\Big)\Big(b(v)h(v)p(v) + \mathcal{PG}(v)\Big). \end{aligned} \tag{26}$$

Finally, to account for the presence of large and small multiplexers in FPGA switch matrices, we further improve the computation of $\mathcal{PG}(v)$:

$$\mathcal{PG}(v) = \begin{cases} 0, & \text{if cluster is in use,} \\ b(v) \cdot (N_L \cdot \mathcal{M} + N_S) \cdot i\,, & \text{otherwise.} \end{cases} \tag{27}$$

Here, $\mathcal{M}$ is the ratio of the power consumption of a large multiplexer over the power consumption of a small one, which we compute in HSPICE simulation. N_L and N_S are the number of large and small multiplexers in the power-gating cluster of the multiplexer v, respectively. Our proposed routing algorithm is different from the previous studies [5], [6] as we precisely consider the architecture of modern FPGAs with various sizes of routing multiplexers. In addition, in contrast to [6], we do not scale down the routing cost with the number of used resources in a power gating region. This is due to the fact that utilizing only one multiplexer in a power gating region forces it to be powered ON and the number of the used multiplexer in a power gating region does not affect its efficiency for power gating. Our scaling of the routing cost with the routing iteration count forces the routing algorithm to avoid multiplexers in unused power gating regions as long as that is possible.

We implement the described enhancement directly in VTR 8.0 and discuss the experimental results in the next section.

V. EXPERIMENTAL SETUP AND RESULTS

In this section, we first detail the implementation and evaluation flow including the associated toolsets, the architectural parameters, and the used benchmark suites. Afterwards, we present the evaluation of the effectiveness of our proposed power gating architectures and the enhanced routing algorithm. Lastly, we provide a comprehensive comparison of our proposed approaches and the closely related research work [5]–[7].

TABLE 4. Benchmarks: The total number of reconfigurable blocks and DSPs is as reported in the work by Murray et al. [28]. The FPGA size is obtained by running the placement and routing of the benchmarks in VTR 8.0, using the Stratix-IV FPGA architectural model.

No.	Name	# Blocks	DSPs	FPGA Size	EXP1	EXP2	EXP3	Application
B1	sparcT1_chip2	814,799	24	279 × 207	L	L	L	Multi-coreμP
B2	LU_Network	630,212	896	221 × 164	L	L	L	Matrix Decomposition
B3	mes_noc	548,047	0	192 × 142	L	L	L	On-chip Network
B4	gsm_switch	487,454	0	255 × 189	L	L	L	Communication Switch
B5	denoise	343,263	192	150 × 111	L	T	L	Image Processing
B6	sparcT2_core	287,839	0	152 × 113	L	T	L	μP Core
B7	cholesky_bdti	257,750	1,027	169 × 125	L	T	T	Matrix Decomposition
B8	minres	252,600	614	224 × 166	L	T	L	Control Systems
B9	stap_qrd	237,193	579	158 × 117	L	L	L	Radar Processing
B10	openCV	212,616	740	232 × 172	L	T	L	Computer Vision
B11	dart	202,414	0	138 × 102	L	L	T	Network Simulator
B12	bitonic_mesh	192,648	676	242 × 179	L	L	T	Sorting
B13	des90	109,962	352	171 × 127	L	L	T	MultiμP system
B14	neuron	90,779	565	129 × 96	L	L	T	Neural Network
B15	segmentation	174,072	104	136 × 101	T	L	T	Computer Vision
B16	SLAM_spheric	124,648	296	124 × 92	T	L	T	Control Systems
B17	cholesky_mc	108,239	452	125 × 93	T	L	T	Matrix Decomposition
B18	stereo_vision	92,662	152	129 × 96	T	L	T	Image Processing
B19	sparcT1_core	91,235	8	82 × 61	T	L	T	μP Core

A. EXPERIMENTAL SETUP

To evaluate the power-gating approaches, we select the Titan benchmarks [28]. These benchmark circuits cover a wide range of application domains. They represent very large industrial-scale designs, many of which contain heterogeneous blocks common in modern FPGAs. In Table 4, we list the benchmark names, the number of reconfigurable blocks they occupy, the number of DSP blocks in use, and the minimal FPGA size needed to place and route them successfully. Additionally, Table 4 shows how we partition the benchmarks into learning (L) and testing (T) sets in three experiments (EXP1, EXP2, and EXP3). In each experiment, the learning benchmarks are used to determine the power-gating clusters, whereas the test benchmarks are used to evaluate their efficiency.

To evaluate the efficiency of our power-gating approaches, we use the Stratix-IV FPGA architecture model—the most advanced FPGA architecture model provided as part of the latest Verilog-to-Routing tool (VTR 8.0) [27], [37]. This FPGA model contains heterogeneous routing and logic resources, similar to that of commercial FPGA devices. The details of the Stratix-IV FPGA architecture model were presented in Section III. In VTR, we set the channel width to $W = 300$, to provide enough routing resources for the largest of the benchmarks. We use COFFE [29] fed with the 22 nm *Predictive Technology Model* (PTM) [59] to automatically generate the transistor sizing, which we then import to HSPICE to estimate the area, delay, and power consumption of the various FPGA resources.

B. EXPERIMENTAL RESULTS

In the following subsections, we present a number of experiments in which we assess the performance of our power-gating methods, by comparing them among themselves and against the approaches proposed by other researchers. Our first comparison metric is the number of routing multiplexers that can be switched off (Section V-B1). Then, we examine the effectiveness of our enhanced router in increasing this particular metric (Section V-B2). Next, we compute the second comparison metric: the area overhead of the power-gating logic (Section V-B3). Afterwards, we focus and elaborate on analyzing the area overhead versus the static power consumption of the FPGA routing resources. Finally, we provide a detailed comparison of our two best-performing methods: *SiM-IPR* and *SiM-IPR-MP* (Section V-B4).

1) How many routing muxes can be switched off?

The most straightforward way to compare our clustering algorithms is to count the number of routing multiplexers that can be switched off (powered off). Table 5 lists the obtained results, for every test benchmark in the three experiments EXP1, EXP2, and EXP3. To facilitate the comparison, the results are normalized with respect to the number of multiplexers that can be switched off when K-means clustering is used instead. In all experiments, we vary the number of clusters K, by setting it to 4, 8, 16, 24, or 32.

The results in Table 5 show that every subsequent proposed strategy resulted in increased (and thus improved) number of multiplexers that can be switched off. Next, for low number of clusters (4, 8, 16), we see that K-means algorithm is, on average, superior than *SiM*, *SiM-PR*, and *SiM-IPR*. For higher number of clusters, however, *SiM-IPR* and *SiM-IPR-MP* take over. Finally, *SiM-IPR-MP* clustering algorithm is the most efficient: For all values of K, *SiM-IPR-MP* outperforms all of the *SiM*, *SiMPR*, *SiM-IPR* approaches, on average. Furthermore, for 32 clusters per switch matrix, *SiM-IPR-MP* algorithm allows for 28% more, on average, multiplexers that can be switched off, compared to the baseline K-means clustering.

Let us now compare *SiM-IPR-MP* approach to the heuristics presented in previous studies, in particular those of Bsoul et al. [5] ($K = 1$), Hoo et al. [6] (two versions: $K = 4$ and $K = 8$[5]), and Seifoori et al. [7] (K = 32, clusters the

[5]All multiplexers in each side are clustered in one power gating group for K = 4, while small multiplexers *12:1* and big multiplexers *40:1* in each side of switch matrices are clustered in different power gating groups in K = 8.

TABLE 5. Number of muxes that can be switched off for 4, 8, 16, 24, and 32 clusters per switch matrix, normalized to those of K-means algorithm. It can be noticed that for low number of clusters (4,8,16), K-means is superior, whereas for higher number of clusters (24, 32) *SiM-IPR-MP* takes over, often outperforming *K*-means by 10–20%.

Experiment-	***SiM***					***SiM-PR***					***SiM-IPR***					***SiM-IPR-MP***				
Benchmark	$K = 32$	$K = 24$	$K = 16$	$K = 8$	$K = 4$	$K = 32$	$K = 24$	$K = 16$	$K = 8$	$K = 4$	$K = 32$	$K = 24$	$K = 16$	$K = 8$	$K = 4$	$K = 32$	$K = 24$	$K = 16$	$K = 8$	$K = 4$
EXP1-B15	1.15	1.09	0.96	0.87	0.82	1.18	1.10	0.97	0.86	0.82	1.18	1.11	0.98	0.87	0.82	1.32	1.26	1.14	1.07	1.04
EXP1-B16	1.07	1.04	0.97	0.9	0.86	1.09	1.04	0.97	0.88	0.86	1.09	1.05	0.98	0.9	0.86	1.22	1.19	1.13	1.09	1.07
EXP1-B17	1.20	1.15	0.03	0.92	0.85	1.21	1.16	1.03	0.89	0.84	1.22	1.16	1.03	0.87	1.07	1.36	1.32	1.2	1.11	1.06
EXP1-B18	1.12	1.1	1.02	0.90	0.83	1.12	1.09	1.02	0.86	0.82	1.13	0.99	1.02	0.88	0.8	1.27	1.25	1.18	1.12	1.07
EXP1-B19	1.09	1.02	0.88	0.76	0.62	1.13	1.02	0.89	0.77	0.60	1.14	1.04	0.92	0.84	0.64	1.25	1.17	1.09	0.97	0.9
EXP2-B5	1.10	0.91	0.69	0.68	0.47	1.12	0.95	0.72	0.75	0.55	1.18	1.00	0.82	0.90	0.50	1.36	1.11	0.92	0.74	0.85
EXP2-B6	1.01	0.90	0.78	0.82	0.62	1.06	0.95	0.88	0.96	0.65	1.03	0.92	0.80	0.86	0.67	1.2	1.03	0.96	0.90	0.85
EXP2-B7	1.2	1.11	1.00	0.92	0.85	1.19	1.12	1.00	0.94	0.86	1.21	1.12	1.00	0.92	0.87	1.39	1.28	1.18	1.09	1.07
EXP2-B8	1.23	1.17	1.05	0.88	0.72	1.23	1.17	1.05	0.89	0.73	1.23	1.17	1.03	0.87	0.75	1.41	1.34	1.26	1.14	1.07
EXP2-B10	1.12	1.07	0.99	0.88	0.80	1.12	1.07	0.99	0.89	0.81	1.13	1.08	0.99	0.89	0.83	1.29	1.23	1.17	1.09	1.07
EXP3-B7	1.06	0.81	0.91	0.79	0.75	1.07	1.02	0.91	0.84	0.79	1.16	1.06	0.96	0.92	0.85	1.28	1.23	1.12	1.06	1.02
EXP3-B11	1.02	0.97	0.91	0.74	0.68	1.03	0.99	0.89	0.81	0.73	1.09	1.02	0.95	0.90	0.81	1.18	1.14	1.08	1.05	1.03
EXP3-B12	1.01	0.94	0.83	0.59	0.49	1.04	0.97	0.82	0.66	0.57	1.14	1.03	0.91	0.85	0.68	1.31	1.28	1.19	1.11	1.00
EXP3-B13	1.00	0.93	1.26	0.54	0.40	1.03	0.96	0.79	0.62	0.49	1.14	1.02	0.90	0.82	0.62	1.32	1.28	1.18	1.10	0.98
EXP3-B14	0.98	0.92	0.83	0.57	0.49	1.01	0.95	0.82	0.68	0.57	1.10	1.00	0.90	0.84	0.7	1.25	1.22	1.15	1.08	1.02
EXP3-B15	1.05	0.98	0.87	0.73	0.69	1.07	1.01	0.88	0.79	0.73	1.16	1.05	0.94	0.88	0.77	1.28	1.21	1.10	1.05	0.99
EXP3-B16	1.00	0.95	0.90	0.74	0.69	1.02	0.98	0.89	0.82	0.75	1.08	1.01	0.94	0.90	0.83	1.18	1.14	1.09	1.06	1.02
EXP3-B17	1.05	1.00	0.90	0.73	0.67	1.07	1.02	0.90	0.79	0.73	1.17	1.06	0.97	0.90	0.82	1.31	1.26	1.16	1.08	1.02
EXP3-B18	0.98	0.93	0.87	0.67	0.62	1.01	0.97	0.86	0.76	0.68	1.09	1.01	0.93	0.87	0.79	1.22	1.19	1.13	1.09	1.02
EXP3-B19	1.06	0.99	0.86	0.71	0.57	1.07	1.02	0.88	0.76	0.64	1.15	1.03	0.92	0.80	0.68	1.22	1.13	1.03	0.96	0.9
Geomean	**1.07**	**0.99**	0.92	**0.76**	**0.66**	**1.09**	**1.03**	**0.90**	**0.81**	**0.7**	**1.14**	**1.04**	**0.94**	**0.87**	**0.76**	**1.28**	**1.21**	**1.12**	**1.04**	**1.00**

TABLE 6. The percentage of all multiplexers that can be switched off using the power-gating schemes proposed in related research works [5]–[7] versus our best performing clustering algorithm *SiM-IPR-MP*.

Experiment-Benchmark	**Bsoul et al. [5]** $K = 1(\%)$	**Hoo et al. [6]** $K = 4(\%)$	***SiM-IPR-MP*** $K = 4(\%)$ (average of 10 runs)	**Hoo et al. [6]** $K = 8(\%)$	***SiM-IPR-MP*** $K = 8(\%)$ (average of 10 runs)	**Seifoori et al. [7]** $K = 32(\%)$	***SiM-IPR-MP*** $K = 32(\%)$ (average of 10 runs)	***Enhanced routing*** $K = 32(\%)$ (average of 10 runs)
EXP1-B15	8.70	14.40	18.48	16.76	24.62	31.09	48.31	55.78
EXP1-B16	15.28	22.66	30.10	24.66	35.69	35.62	53.04	57.50
EXP1-B17	10.24	17.31	22.85	19.50	30.28	36.41	57.01	62.27
EXP1-B18	19.17	33.97	45.32	37.11	56.48	51.48	82.06	85.97
EXP1-B19	0.35	2.42	2.71	4.03	5.31	12.61	19.31	24.30
EXP2-B5	0.19	1.97	1.73	4.34	3.64	14.86	21.74	32.45
EXP2-B6	0.13	1.35	1.29	2.74	2.64	9.23	13.81	18.17
EXP2-B7	11.46	15.99	21.02	18.14	26.89	33.39	52.01	57.04
EXP2-B8	6.54	21.22	27.12	24.32	39.53	44.10	69.90	73.93
EXP2-B10	17.58	28.55	36.65	31.37	45.79	43.44	69.51	71.62
EXP3-B7	11.46	15.99	19.96	18.14	25.84	33.39	49.82	54.56
EXP3-B11	12.27	17.97	22.22	19.86	26.88	28.12	42.38	46.85
EXP3-B12	6.17	18.48	22.04	21.69	33.46	40.44	61.43	64.19
EXP3-B13	2.97	15.86	18.30	19.03	29.38	37.73	57.34	60.44
EXP3-B14	10.36	28.17	34.24	31.38	45.98	47.59	72.41	76.71
EXP3-B15	8.70	14.41	17.26	16.76	23.55	31.09	46.32	53.89
EXP3-B16	15.28	22.66	28.13	24.66	34.44	35.62	50.76	54.84
EXP3-B17	10.24	17.31	21.28	19.50	29.04	36.41	54.59	59.76
EXP3-B18	19.17	33.97	42.47	37.11	54.13	51.48	77.34	81.40
EXP3-B19	0.35	2.42	2.47	4.03	5.01	12.61	18.49	23.23
Geomean	**4.93**	**12.76**	**15.28**	**15.94**	**22.07**	**30.24**	**45.92**	**51.74**

multiplexers driving the same track in the same power gating groups). Table 6 reports how many (in %) of all routing multiplexers can be turned off, for all the test benchmarks and in all three experiments. Since the pattern of some clusters in each clustering iteration of *SiM-IPR-MP* algorithm is replaced with a randomly selected member of the cluster, the achieved clustering varies randomly from run to run. To investigate how this randomness would affect the number of multiplexers that can be powered off, we repeat the *SiM-IPR-MP* clustering algorithm for each benchmark 10 times. In the last column of Table 6, we add the values obtained when our enhanced router is used after the clustering with *SiM-IPR-MP*, which also has been repeated 10 times. The average of the standard deviation across 10 runs over all benchmarks in *SiM-IPR-MP* clustering algorithm for four, eight, and 32 clusters per switch matrix is 0.36%, 0.25%, and 0.2%, respectively. Furthermore, the maximum standard deviation is 1.25%, 0.81%, and 2.38%, respectively. The average of the standard deviation across 10 runs over all benchmarks for enhanced router is 0.61%, while the maximum is 1.18%. This indicates that there is good agreement over the repeated simulation runs and suggests that running simulation for only one time can effectively verify the proposed algorithms. Looking at the average results, we see the following: For four clusters, *SiM-IPR-MP* can power off additional 20% of the routing muxes (15.28% vs. 12.76%). For eight clusters, it can power off additional 38% multiplexers (22.07% vs. 15.94%) and, for 32 clusters, this number increases to 52% multiplexers (45.92% vs. 30.24%). Finally, the enhanced routing helps power off additional 19.3% of all the routing multiplexers, amounting to ≈71% improvement (51.74% vs. 30.24%) compared to the best static power-gating approach reported so far.

TABLE 7. Number of muxes that can be switched off when the enhanced routing algorithm and *SiM-IPR-MP* clustering are used, for 16, 24, and 32 clusters per switch matrix, normalized to the corresponding number of muxes in the absence of the enhanced routing algorithm. The results show that the enhanced routing algorithm helps increasing the power gating opportunities by 10–18%, on average.

Experiment-	*Power-Gating-Aware Routing*		
Benchmark	$K = 32$	$K = 24$	$K = 16$
EXP1-B15	1.16	1.26	1.24
EXP1-B16	1.09	1.14	1.10
EXP1-B17	1.10	1.17	1.15
EXP1-B18	1.05	1.09	1.10
EXP1-B19	1.25	1.44	-
EXP2-B5	1.45	1.76	2.02
EXP2-B6	1.30	1.42	-
EXP2-B7	1.08	1.14	1.16
EXP2-B8	1.05	1.08	1.12
EXP2-B10	1.02	1.04	1.07
EXP3-B7	1.04	1.02	1.11
EXP3-B11	1.26	1.01	-
EXP3-B12	1.09	1.03	1.11
EXP3-B13	1.16	1.03	1.25
EXP3-B14	1.05	1.01	1.14
EXP3-B15	1.04	1.08	1.14
EXP3-B16	1.11	1.12	1.15
EXP3-B17	1.09	1.13	1.17
EXP3-B18	1.09	0.83	1.19
EXP3-B19	1.04	0.98	1.13
Geomean	**1.12**	**1.12**	**1.18**

TABLE 8. Critical path delay when the enhanced routing algorithm is used, for 16, 24, and 32 clusters, normalized with the critical path delay obtained using the unmodified VTR router.

Experiment-	*Power-Gating-Aware Routing*		
Benchmark	$K = 32$	$K = 24$	$K = 16$
EXP1-B15	0.99	1.00	0.99
EXP1-B16	0.99	1.00	1.00
EXP1-B17	1.13	1.00	1.50
EXP1-B18	0.99	0.92	0.98
EXP1-B19	1.13	0.85	-
EXP2-B5	0.99	0.99	0.99
EXP2-B6	1.08	1.12	-
EXP2-B7	1.08	1.17	1.09
EXP2-B8	1.19	1.52	1.40
EXP2-B10	1.18	1.23	1.29
EXP3-B7	0.99	1.00	1.11
EXP3-B11	1.18	1.04	-
EXP3-B12	0.99	1.01	1.00
EXP3-B13	0.99	1.01	1.00
EXP3-B14	0.99	1.00	1.06
EXP3-B15	1.26	1.14	1.24
EXP3-B16	1.09	1.14	1.19
EXP3-B17	1.10	1.47	1.29
EXP3-B18	1.23	1.25	1.36
EXP3-B19	1.26	1.21	1.18
Geomean	**1.09**	**1.09**	**1.15**

2) Effectiveness of the power-gating-aware router

Table 7 shows the effectiveness of power-gating-aware routing algorithm in increasing the power gating opportunities. We test this routing algorithm for *SiM-IPR-MP* clustering and the number of clusters set to 16, 24, and 32. Reducing the number of clusters below 16 results in higher number of multiplexers in each power gating cluster. As Eq. (27) suggests, with the increase of the number of multiplexers per cluster, the routing resource cost $Cost_{PG}$ increases as well. When this cost is too high, the routing fails, which is why the enhanced routing algorithm fails to successfully route the test benchmarks for four and eight clusters per switch matrix, as well as the three benchmarks indicated with a dash in Table 7. In almost all the remaining experiments, we found that the enhanced router helps to use better the power-gating regions: on average, the number of additional muxes that could be switched off reaches 12% (for $K = 32$), 12% (for $K = 24$), and 18% (for $K = 16$); this increase is comparable to the improvement brought by *SiM-IPR-MP* over *SiM-IPR*.

When evaluating an FPGA router, it is common to report its impact on the circuit critical path delay. We show this data in Table 8; on average, the critical path delay increases by 9% ($K = 32$ and $K = 24$) or 15% ($K = 16$). This increase is due to the router putting more effort in saving power than in finding the most efficient routes. When the power consumption is the main design concern, this increase can probably be tolerated.

TABLE 9. The area overhead in the FPGA routing switch matrices after adding the power-gating logic, for *SiM-IPR* clustering algorithm.

SiM-IPR $K = 32$	*SiM-IPR* $K = 24$	*SiM-IPR* $K = 16$	*SiM-IPR* $K = 8$	*SiM-IPR* $K = 4$
5.48%	4.12%	2.69%	1.18%	0.58%

TABLE 10. The area overhead in the FPGA routing switch matrices after adding the power-gating logic, for *SIM-IPR-MP* clustering algorithm.

SiM-IPR-MP $K = 32$	*SiM-IPR-MP* $K = 24$	*SiM-IPR-MP* $K = 16$	*SiM-IPR-MP* $K = 8$	*SiM-IPR-MP* $K = 4$
5.61%	4.21%	2.95%	1.23%	0.64%

TABLE 11. The area overhead in the routing switch matrices after adding the power-gating logic, computed using the clustering approaches proposed by other researchers.

Bsoul et al. [5] $K = 1$	**Hoo et al. [6]** $K = 4$	**Hoo et al. [6]** $K = 8$	**Seifoori et al. [7]** $K = 32$
0.136%	0.55%	0.88%	6.16%

3) Area overhead of the power-gating logic

Let us now compare the area overhead of our power-gating architectures, which we define as the difference between the area required for the routing switch matrix with and without the power-gating circuitry. For this purpose, we perform HSPICE circuit-level simulations, using the accurate netlists of the FPGA power-gating regions generated by COFFE, and 22 nm predictive technology model [59].

Table 9 and Table 10 show the overhead obtained for *SiM-IPR* and *SiM-IPR-MP* clustering approaches, averaged across all benchmarks and all three experiments. Table 11 shows the area overhead obtained for the power-gating approaches proposed by other researchers. It is not surprising that the power overhead grows with the increase in the number of clusters per switch matrix K, because the number of power-gating (*sleep*) transistors is proportional to the number of clusters per switch matrix. For all values of K, we measure similar area overhead compared to the respective related work.

The slight differences are due to the fact that the area overhead depends on the cluster size and its composition: our clustering, unlike the approaches proposed by other researchers, results in nonuniform cluster sizes, which is favorable when the number of clusters is high ($K = 32$).

Fig. 11 shows the power-gating area overhead (y-axis) versus the achieved *normalized* static power consumption (x-axis), for all test benchmarks and all three experiments, selecting the following power-gating approaches: Bsoul et al. [5] ($K = 1$), Hoo et al. [6] (two versions: $K = 4$ and $K = 8$), Seifoori et al. [7] ($K = 32$), *KM*, *SiM*, and *SiM-IPR*. The values on x-axis are normalized to the static power consumption estimated for an FPGA architecture without any of the power-gating mechanisms in place. This figure is in line with data presented in Table 9, Table 10, and Table 11: our approaches result in area overhead higher than that of Bsoul et al. [5] ($K = 1$) and Hoo et al. [6] ($K = 4$ and $K = 8$), but lower than that of Seifoori et al. [7]. Additionally, our approaches result in all data points being shifted towards lower normalized static power consumption (lower values on x-axis), which is, again, in line with our design goals. Dashed line emphasizes the best results achieved by our algorithms, which outperform all the other power-gating strategies.

Fig. 12 compares *SiM-IPR-MP*, with and without the enhanced routing, against the *SiM-IPR* approach. Each color in the plot corresponds to a specific cluster size K. We notice that as K grows, the area overhead grows as well; the reason for this is the increased number of power-gating logic that needs to be put in place. However, higher K has important benefits: it helps reducing the normalized static power consumption (the data points move to the left on x-axis). The square symbols correspond to the enhanced router results—as expected, they form the Pareto front. The normalized power consumption of data points in Fig. 11 and Fig.12 is reported in Table 12. Similar to Table 6, the normalized power consumption of data points reported in Table 12 using *SiM-IPR-MP* algorithm and enhanced router algorithm is the average across 10 runs over all benchmarks. We see that employing *SiM-IPR-MP* algorithm with 32 clusters per switch matrix and the power-gating aware router can shrink the FPGA static power consumption by 53%, on average, whereas the best power-gating approach presented by Seifoori et al. [7] achieves 31% power reduction, on average; this presents an improvement of about 70% over the state of the art.

4) How well SiM-IPR-MP groups the big muxes?

As discussed in Section IV-B4, one of the goals of *SiM-IPR-MP* clustering algorithm is to improve the clustering of big multiplexers, so that a higher number of them can be powered off. To estimate how well *SiM-IPR-MP* performs this task, we first count the number of big multiplexers that can be powered off and compare it to *SiM-IPR*. Then, we introduce a metric called *power-reduction rate* (*PR*), defined as

$$PR = \frac{P_{IPR} - P_{MP}}{P_{IPR}}. \tag{28}$$

Here, P_{IPR} and P_{MP} are the estimated power consumption when *SiM-IPR* and *SiM-IPR-MP* clustering approaches are used, respectively.

In Fig. 13, for all the test benchmarks and all three experiments, we plot on y-axis the relative change in the number of powered-off big multiplexers (the higher, the better) and on x-axis the power reduction rate (in %). These results demonstrate that *SiM-IPR-MP* clustering algorithm does manage to improve the clustering of big multiplexers by outperforming *SiM-IPR* approach in almost all test cases. In the couple of encircled outlier cases, even though the number of powered-off multiplexers worsens, the actual power saving is improved, thanks to the comparatively higher number of powered-off small multiplexers.

Fig. 14 provides another comparison of *SiM-IPR-MP* and *SiM-IPR* algorithms. Each marker corresponds to one experiment-benchmark pair. On x-axis, we show the power-reduction rate, corresponding to Eq. (28). On y-axis, we choose to show another metric: *Power-Overhead-reduction Rate* (*POR*). It is computed as follows:

$$POR = \frac{PO_{IPR} - PO_{MP}}{PO_{IPR}}. \tag{29}$$

Here, the power overhead *PO* is the difference between, on the one hand, the sum of the power consumption of all the powered-off regions and all the powered-on regions and, on the other hand, the sum of the power consumption of all the routing multiplexers when no power gating is put in place. Therefore, the higher the value on y-axis, the higher the power-overhead reduction, i.e., the lower the power overhead. Analyzing the results in Fig. 14, we see that for almost all test cases, *SiM-IPR-MP* is superior to *SiM-IPR*: the majority of the data points are situated in the first quadrant. The highest power-overhead reduction and the highest decrease in the overall power consumption are both $\approx 25\%$. There are, however, a few encircled outliers, for which either the power overhead or the overall power consumption is slightly increased compared to *SiM-IPR*, due to the final cluster composition and the number of small and big multiplexers in each cluster.

Fig. 15 shows the average of the power reduction rate of data points in Fig. 14 versus the average of their normalized power overhead. The y-axis on the left corresponds to the average power-reduction rate and the y-axis on the right corresponds the average of power overhead, which are normalized to the routing power consumption of each benchmark implemented in an architecture that does not support power gating, for the sake of fair comparison. We can observe that *SiM-IPR-MP* is superior in both the power-reduction rate and the power-overhead. Additionally, 32 clusters per switch matrix is once again the optimum configuration, with both metrics reaching their high values.

VI. CONCLUSION

This paper discussed the effectiveness of leveraging machine-learning approaches and power-gating aware routing in reducing the static power consumption of FPGA routing resources.

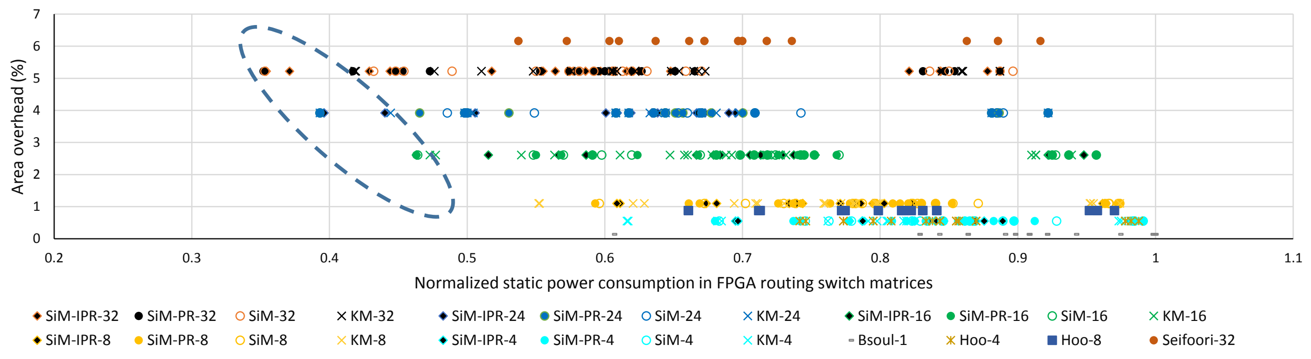

FIGURE 11. The power-gating area overhead versus the static power consumption of the FPGA routing network. The latter is normalized to the static power consumption estimated for an FPGA architecture without any of the power-gating mechanisms in place. Each marker corresponds to one experiment-benchmark pair.

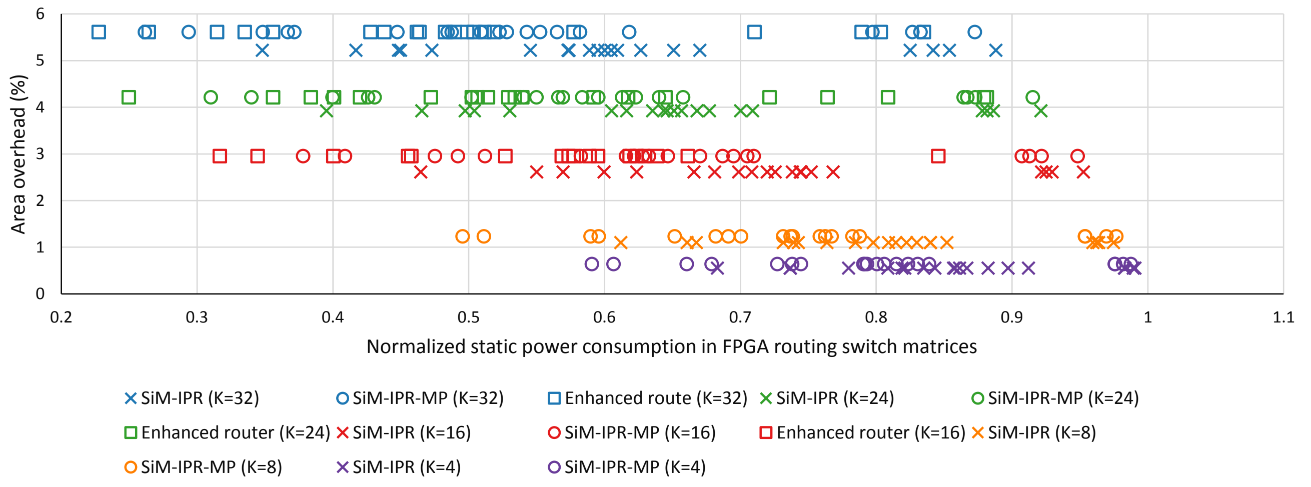

FIGURE 12. The power-gating area overhead versus the static power consumption for *SiM-IPR-MP* and *SiM-IPR* clustering. The latter is normalized to the static power consumption for an FPGA architecture without any of the power-gating mechanisms in place. Each marker corresponds to one experiment-benchmark pair.

TABLE 12. The static power consumption of the FPGA routing network in FPAG architectures that use the power gating schemes proposed in previous studies [5]–[7] or our best performing clustering algorithm *SiM-IPR-MP*. The power consumption is normalized to the static power consumption of an FPGA architecture which does not employ any power gating scheme.

Experiment-Benchmark	Bsoul et al. [5] $K=1$	Hoo et al. [6] $K=4$	*SiM-IPR-MP* $K=4$ (average of 10 runs)	Hoo et al. [6] $K=8$	*SiM-IPR-MP* $K=8$ (average of 10 runs)	Seifoori et al. [7] $K=32$	*SiM-IPR-MP* $K=32$ (average of 10 runs)	*Enhanced routing* $K=32$ (average of 10 runs)
EXP1-B15	0.92	0.87	0.83	0.84	0.78	0.72	0.56	0.50
EXP1-B16	0.86	0.79	0.73	0.77	0.68	0.69	0.52	0.48
EXP1-B17	0.91	0.84	0.79	0.82	0.73	0.67	0.49	0.44
EXP1-B18	0.83	0.77	0.59	0.66	0.49	0.54	0.26	0.23
EXP1-B19	0.99	0.98	0.98	0.96	0.95	0.88	0.82	0.78
EXP2-B5	0.99	0.98	0.98	0.95	0.97	0.86	0.80	0.71
EXP2-B6	0.99	0.98	0.99	0.97	0.98	0.92	0.87	0.84
EXP2-B7	0.89	0.85	0.81	0.83	0.76	0.69	0.53	0.49
EXP2-B8	0.94	0.81	0.76	0.77	0.64	0.60	0.37	0.33
EXP2-B10	0.84	0.74	0.67	0.71	0.59	0.61	0.37	0.35
EXP3-B7	0.89	0.85	0.82	0.83	0.77	0.69	0.55	0.51
EXP3-B11	0.88	0.84	0.80	0.81	0.76	0.74	0.62	0.58
EXP3-B12	0.61	0.83	0.80	0.79	0.70	0.64	0.45	0.42
EXP3-B13	0.97	0.86	0.83	0.82	0.74	0.66	0.48	0.46
EXP3-B14	0.91	0.75	0.69	0.71	0.59	0.57	0.35	0.31
EXP3-B15	0.92	0.87	0.84	0.84	0.79	0.71	0.58	0.51
EXP3-B16	0.86	0.79	0.75	0.77	0.69	0.69	0.54	0.51
EXP3-B17	0.91	0.84	0.81	0.82	0.74	0.67	0.51	0.46
EXP3-B18	0.83	0.77	0.62	0.66	0.51	0.54	0.30	0.27
EXP3-B19	0.99	0.98	0.98	0.96	0.95	0.88	0.83	0.79
Geomean	**0.89**	**0.85**	**0.80**	**0.81**	**0.73**	**0.69**	**0.51**	**0.47**

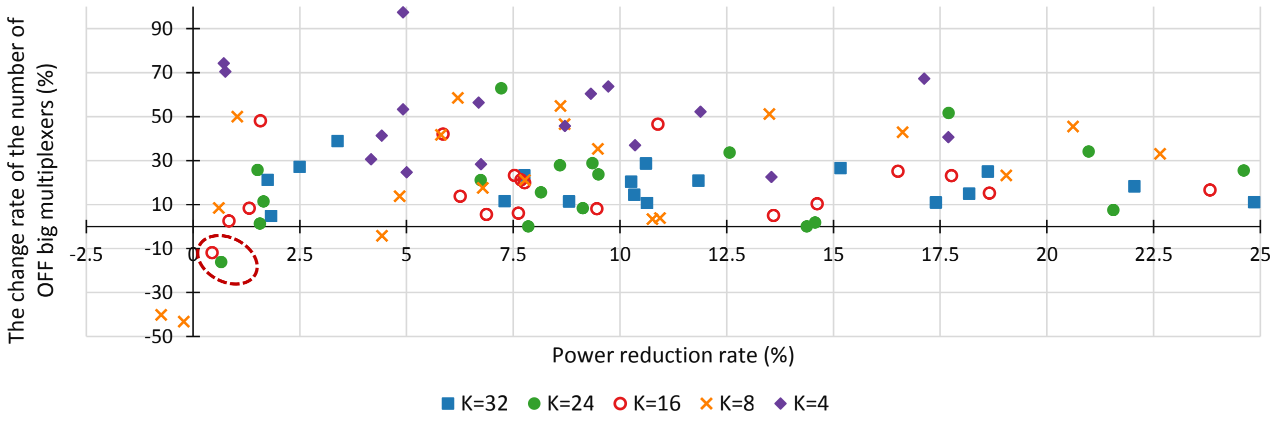

FIGURE 13. The power reduction rate in Eq. (28) versus the change rate of the number of switched OFF big multiplexers for *SiM-IPR-MP* clustering algorithm versus with *SiM-IPR*, in three experiments of EXP1, EXP2, EXP3 and for various number of clusters. Each marker correspond to a specific number of clusters.

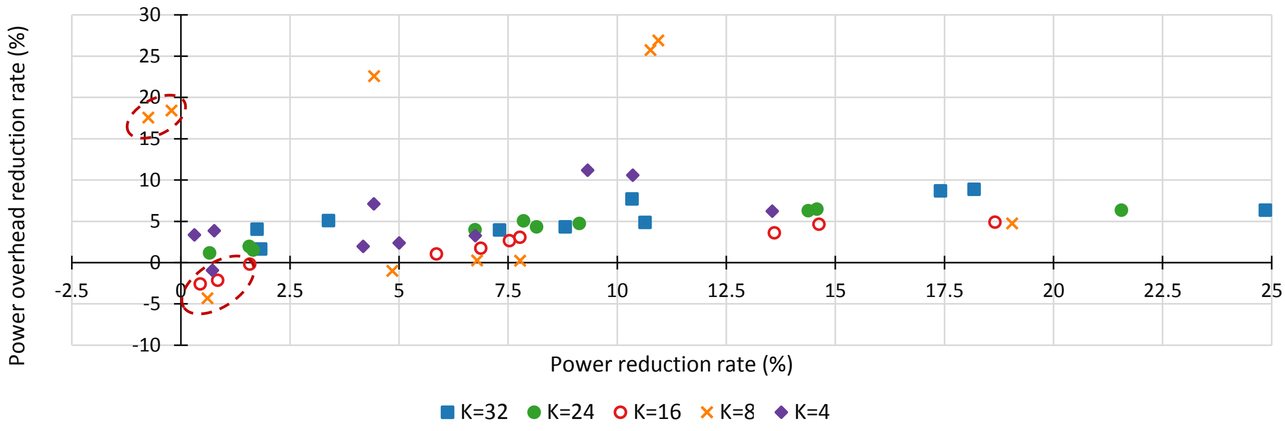

FIGURE 14. The power reduction rate in Eq. (28) versus the power consumption overhead reduction rate in Eq. (29), for all three experiments EXP1, EXP2, and EXP3. This figure illustrates the improvement brought by *SiM-IPR-MP* clustering algorithm over *SiM-IPR*.

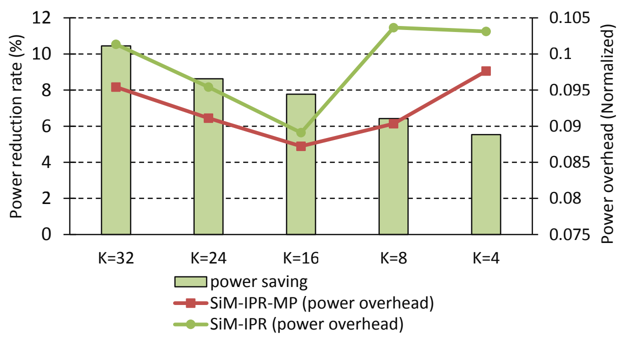

FIGURE 15. The average of power-reduction rate and normalized power-overhead, in the function of cluster size, for *SiM-IPR-MP* and *SiM-IPR* clustering.

To improve the efficiency of previously proposed power-gating techniques, which all relied on heuristic approaches, we proposed a number of machine-learning inspired clustering algorithms, each more efficient than previous: *SiM*, *SiM-PR*, *SiM-IPR*, and *SiM-IPR-MP*. Additionally, we design an improved FPGA router, which takes into account the availability of power-gating regions and further helps their efficient use. The experimental results are very promising: in a setting with 32 clusters per switch matrix, our most efficient clustering algorithm *SiM-IPR-MP*—which takes into account the power overhead of the power-gating circuitry and the routing multiplexer size—helps reducing the static power consumption by additional ≈26% (0.51 vs. 0.69), on average, with respect to the most efficient similar solution proposed so far [7]. When power-gating aware routing is applied together with *SiM-IPR-MP* clustering, we observe additional ≈4% reduction in static power consumption, on average. This work is, therefore, an important step towards enabling wider presence of FPGAs in low-power applications.

REFERENCES

[1] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, pp. 203–215, Jan. 2007.
[2] Y. Lin, F. Li, and L. He, “Routing track duplication with fine-grained

power-gating for FPGA interconnect power reduction," in Proceedings of the Asia and South Pacific design automation conference, Shanghai China, Jul. 2005, pp. 645–650.

[3] A. A. Bsoul and S. J. Wilton, "An FPGA architecture supporting dynamically controlled power gating," in Proceeding of International Conference on Field-Programmable Technology (FPT), Beijing, China, Jan. 2010, pp. 1–8.

[4] A. Gayasen, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "Reducing leakage energy in FPGAs using region-constrained placement," in Proceedings of the 12th international symposium on field programmable gate arrays, Monterey California USA, Feb. 2004, pp. 51–58.

[5] A. A. Bsoul and S. J. Wilton, "An FPGA with power-gated switch blocks," in Proceedings of International Conference on Field-Programmable Technology (FPT), Seoul, South Korea, Jan. 2012, pp. 87–94.

[6] C. H. Hoo, Y. Ha, and A. Kumar, "A directional coarse-grained power gated FPGA switch box and power gating aware routing algorithm," in Proceedings of 23rd International Conference on Field Programmable Logic and Applications (FPL). Porto, Portugal: IEEE, Oct. 2013, pp. 1–4.

[7] Z. Seifoori, B. Khaleghi, and H. Asadi, "A power gating switch box architecture in routing network of SRAM-based FPGAs in dark silicon era," in Proceeding of Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, May 2017, pp. 1342–1347.

[8] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA," in Proceedings of the Custom Integrated Circuits Conference, San Jose, CA, USA, USA, Dec. 2003, pp. 57–60.

[9] V. Degalahal and T. Tuan, "Methodology for high level estimation of FPGA power consumption," in Proceedings of the Asia and South Pacific Design Automation Conference, Shanghai China, Jul. 2005, pp. 657–660.

[10] P. Huang, Z. Xing, T. Wang, Q. Wei, H. Wang, and G. Fu, "A brief survey on power gating design," in Proceeding of 10th International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Shanghai, China, Dec. 2010, pp. 788–790.

[11] H. Jiang, M. Marek-Sadowska, and S. R. Nassif, "Benefits and costs of power-gating technique," in International Conference on Computer Design, San Jose, CA, USA, USA, Oct. 2005, pp. 559–566.

[12] A. A. Bsoul, S. J. Wilton, K. H. Tsoi, and W. Luk, "An FPGA architecture and CAD flow supporting dynamically controlled power gating," IEEE Transactions on Very Large Scale Integration VLSI Systems, vol. 24, no. 1, pp. 178–191, Feb. 2016.

[13] A. Ahari, B. Khaleghi, Z. Ebrahimi, H. Asadi, and M. B. Tahoori, "Towards dark silicon era in FPGAs using complementary hard logic design," in Proceeding of 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, Germany, Oct. 2014, pp. 1–6.

[14] S. Ishihara, M. Hariyama, and M. Kameyama, "A low-power FPGA based on autonomous fine-grain power gating," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 8, pp. 1394–1406, Jun. 2011.

[15] Z. Ebrahimi, B. Khaleghi, and H. Asadi, "PEAF: A power-efficient architecture for SRAM-based FPGAs using reconfigurable hard logic design in dark silicon era," IEEE Transactions on Computers, vol. 66, no. 6, pp. 982–995, Dec. 2017.

[16] S. Yazdanshenas and H. Asadi, "Fine-grained architecture in dark silicon era for SRAM-based reconfigurable devices," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 61, no. 10, pp. 798–802, Aug. 2014.

[17] C. Li, Y. Dong, and T. Watanabe, "New power-aware placement for region-based FPGA architecture combined with dynamic power gating by PCHM," in IEEE/ACM International Symposium on Low Power Electronics and Design, Fukuoka, Japan, Aug. 2011, pp. 223–228.

[18] A. Mametjanov, P. Balaprakash, C. Choudary, P. D. Hovland, S. M. Wild, and G. Sabin, "Autotuning FPGA design parameters for performance and power," in 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, Canada, Jul. 2015, pp. 84–91.

[19] N. Kapre, H. Ng, K. Teo, and J. Naude, "Intime: A machine learning approach for efficient selection of FPGA CAD tool parameters," in Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey California USA, Feb. 2015, pp. 23–26.

[20] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A parallel bandit-based approach for autotuning FPGA compilation," in Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey California USA, Feb. 2017, pp. 157–166.

[21] Q. Yanghua, C. Adaikkala Raj, H. Ng, K. Teo, and N. Kapre, "Case for design-specific machine learning in timing closure of FPGA designs," in Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey California USA, Feb. 2016, pp. 169–172.

[22] E. Ustun, S. Xiang, J. Gui, C. Yu, and Z. Zhang, "Lamda: Learning-assisted multi-stage autotuning for FPGA design closure," in 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, USA, Jun. 2019, pp. 74–77.

[23] Z. Qi, Y. Cai, and Q. Zhou, "Accurate prediction of detailed routing congestion using supervised data learning," in 32nd International Conference on Computer Design (ICCD), Seoul, South Korea, Dec. 2014, pp. 97–103.

[24] G. Gréwal, S. Areibi, M. Westrik, Z. Abuowaimer, and B. Zhao, "Automatic flow selection and quality-of-result estimation for FPGA placement," in International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, USA, Jul. 2017, pp. 115–123.

[25] C.-W. Pui, G. Chen, Y. Ma, E. F. Young, and B. Yu, "Clock-aware ultrascale FPGA placement with machine learning routability prediction," in International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, Dec. 2017, pp. 929–936.

[26] J. Zhao, T. Liang, S. Sinha, and W. Zhang, "Machine learning based routing congestion prediction in FPGA high-level synthesis," in Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, Italy, May 2019, pp. 1130–1135.

[27] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. Walker et al., "VTR 8: High-performance cad and customizable FPGA architecture modelling," ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 13, no. 2, pp. 1–55, May 2020.

[28] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal, Oct. 2013, pp. 1–8.

[29] C. Chiasson and V. Betz, "COFFE: Fully-automated transistor sizing for FPGAs," in Proceeding of International Conference on Field-Programmable Technology (FPT), Kyoto, Japan, Jan. 2013, pp. 34–41.

[30] Z. Seifoori, H. Asadi, and M. Stojilović, "A machine learning approach for power gating the FPGA routing network," in International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, China, Feb. 2019, pp. 10–18.

[31] S. Yang, Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0. Microelectronics Center of North Carolina (MCNC), 1991.

[32] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, "Hl-pow: A learning-based power modeling framework for high-level synthesis," in 25th Asia and South Pacific Design Automation Conference (ASP-DAC), Beijing, China, Jan. 2020, pp. 574–580.

[33] H. M. Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M. P. Dinakarrao, H. Homayoun, and S. Rafatirad, "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in 29th International Conference on Field Programmable Logic and Applications (FPL). Barcelona, Spain: IEEE, Sep. 2019, pp. 397–403.

[34] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate operation delay prediction for FPGA HLS using graph neural networks," in Proceedings of the 39th International Conference on Computer-Aided Design, Virtual event USA, Nov. 2020, pp. 1–9.

[35] D. Maarouf, A. Alhyari, Z. Abuowaimer, T. Martin, A. Gunter, G. Grewal, S. Areibi, and A. Vannelli, "Machine-learning based congestion estimation for modern fpgas," in 28th International Conference on Field Programmable Logic and Applications (FPL). Dublin, Ireland: IEEE, Aug. 2018, pp. 427–4277.

[36] "Virtex-6 FPGA configurable logic block." User Guide, Xilinx, February, 2012.

[37] "StratixIV device handbook." Handbook, Altera, January, 2016.

[38] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed et al., "VTR 7.0: Next generation architecture and CAD system for FPGAs," IEEE Transactions on Reconfigurable Technology and Systems TRETS, vol. 7, no. 2, pp. 1–30, Jul. 2014.

[39] (2005 (accessed December 30, 2018)) IWLS 2005 benchmarks. [Online]. Available: http://iwls.org/iwls2005/benchmarks.html

[40] R. O. Duda, P. E. Hart et al., Pattern classification and scene analysis. Wiley New York, 1973, vol. 3.
[41] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Advances in knowledge discovery and data mining. The MIT Press, 1996.
[42] A. K. Jain and R. C. Dubes, Algorithms for clustering data. Prentice-Hall, Inc., 1988.
[43] A. Gersho and R. M. Gray, Vector quantization and signal compression. Springer Science & Business Media, 2012, vol. 159.
[44] R. Xu and D. Wunsch, "Survey of clustering algorithms," IEEE Transactions on Neural Networks, vol. 16, no. 3, pp. 645–678, 2005.
[45] S. Sharma and S. Rai, "Genetic k-means algorithm implementation and analysis," International Journal of Recent Technology and Engineering, vol. 1, no. 2, pp. 117–120, Jun. 2012.
[46] Y.-C. Chiou and L. W. Lan, "Genetic clustering algorithms," European Journal of Operational Research, vol. 135, no. 2, pp. 413–427, Dec. 2001.
[47] P. K. Agarwal and C. M. Procopiuc, "Exact and approximation algorithms for clustering," Algorithmica, vol. 33, no. 2, pp. 201–226, Jan. 2002.
[48] S. Arora, P. Raghavan, and S. Rao, "Approximation schemes for Euclidean k-medians and related problems," in The 30th Annual ACM Symposium on Theory of Computing. Dallas Texas USA: ACM, May 1998, pp. 106–113.
[49] S. G. Kolliopoulos and S. Rao, "A nearly linear-time approximation scheme for the Euclidean k-median problem," (SIAM) Journal on Computing, vol. 37, no. 3, pp. 757–782, Jun. 2007.
[50] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," Pattern Recognition, vol. 36, no. 2, pp. 451–461, Feb. 2003.
[51] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, Tech. Rep., Jan. 2006.
[52] A. K. Jain, "Data clustering: 50 years beyond k-means," Pattern Recognition letters, vol. 31, no. 8, pp. 651–666, Jun. 2010.
[53] Z. Cai, M. Heydari, and G. Lin, "Clustering binary oligonucleotide fingerprint vectors for DNA clone classification analysis," Journal of Combinatorial Optimization, vol. 9, no. 2, pp. 199–211, Mar. 2005.
[54] Z. Cai, R. Goebel, M. R. Salavatipour, Y. Shi, L. Xu, and G. Lin, "Selecting genes with dissimilar discrimination strength for sample class prediction," in Proceedings of the 5th Asia-Pacific Bioinformatics Conference. Hong Kong, China: World Scientific, Jan. 2007, pp. 81–90.
[55] Z. Cai, L. Xu, Y. Shi, M. R. Salavatipour, R. Goebel, and G. Lin, "Using gene clustering to identify discriminatory genes with higher classification accuracy," in Proceedings of Sixth Symposium on BioInformatics and BioEngineering, Arlington, VA, USA, Dec. 2006, pp. 235–242.
[56] S. P. Lloyd, "Least squares quantization in PCM," IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129–137, Mar. 1982.
[57] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in Reconfigurable Computing. Elsevier, May 2008, pp. 365–381.
[58] M. E. Celebi, H. A. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the K-means clustering algorithm," Expert Systems with Applications, vol. 40, no. 1, pp. 200–210, Jan. 2013.
[59] (2013 (accessed November 1, 2020)) Predictive technology model (PTM). [Online]. Available: http://ptm.asu.edu/

ZEINAB SEIFOORI received the B.Sc. and M.Sc. degrees in computer engineering from Shahed University and Sharif University of Technology (SUT), Tehran, Iran, in 2006 and 2010, respectively. From 2015, she has been working toward the Ph.D. degree in computer engineering in Data Storage, Networks, and Processing (DSN) Laboratory at SUT under supervision of Dr. Hossein Asadi. She spent nine months as a research assistant at EPFL School of Computer and Communication Sciences. Her research interests include reconfigurable computing and reliability of computer systems. She was nominated for the Best paper award at 2019 International Conference on Field-Programmable Technology (ICFPT).

HOSSEIN ASADI (M'08, SM'14) received the BSc and MSc degrees in computer engineering from the Sharif University of Technology (SUT), Tehran, Iran, in 2000 and 2002, respectively, and the PhD degree in computer engineering from Northeastern University, Boston, MA, USA, in 2007. He was with EMC Corporation, Hopkinton, MA, as a research scientist and senior hardware engineer from 2006 to 2009. He is currently a full professor in the Department of Computer Engineering, SUT. He is the founder and director of the Data Storage, Networks, and Processing (DSN) Laboratory and the director of Sharif High-Performance Computing (HPC) Center. He is also the co-founder of HPDS corp., designing and fabricating midrange and high-end data storage systems. His current research interests include data storage systems and networks, solid-state drives, operating systems, and high-performance, reconfigurable, and dependable computing.

Dr. Asadi was a recipient of the Technical Award for the Best Robot Design from the International RoboCup Rescue Competition, organized by AAAI and RoboCup in 2001, a recipient of Best Paper Award at the 15th CSI International Symposium on Computer Architecture & Digital Systems (CADS) in 2010, the Distinguished Lecturer Award from SUT in 2010, the Distinguished Researcher Award and the Distinguished Research Institute Award from SUT in 2016, the Distinguished Technology Award from SUT in 2017, and the Distinguished Research Lab Award from SUT in 2019. He has been ranked among "Top-10" among 500+ faculties by Research and Technology Deputy, SUT for five consecutive years from 2016 to 2020. More recently, he received the Best Paper Award at IEEE/ACM Design, Automation, and Test in Europe (DATE) in 2019. He has served as a guest editor of IEEE Transactions on Computers, an Associate Editor of Microelectronics Reliability, a Program Co-Chair of CADS2015, and the Program Chair of CSI National Computer Conference (CSICC2017). He is a senior member of the IEEE.

MIRJANA STOJILOVIĆ (M'09–SM'19) received the Dipl. Ing. and Ph.D. degrees from the School of Electrical Engineering, University of Belgrade, Serbia, in 2006 and 2013, respectively. In 2016, she joined the School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. Her current research interests include electronic design automation, reconfigurable computing, and hardware security.

In 2019, Dr. Stojilović was nominated for the Best Paper Award (BPA) at the International Conference on Field-Programmable Technology (FPT). She was a recipient of the BPA at EMC Europe 2016, the Young Scientist Award at ICLP 2016, and the Young Author BPA at TELFOR 2012. In 2015, the EPFL School of Computer and Communication Sciences presented her with the Teaching Award.

Dr. Stojilović serves on the program committees of FPGA, FPL, and FCCM conference. In 2021, she was on the BPA committee of the FPGA conference. Since 2019, she has been leading the project "Secure FPGAs in the Cloud," funded by the Swiss National Science Foundation.