STT-MRAM-Based Multicontext FPGA for Multithreading Computing Environment

Jeongbin Kim, Yongwoon Song, *Student Member, IEEE*, Kyungseon Cho, Hyukjun Lee, *Member, IEEE*, Hongil Yoon, *Member, IEEE*, and Eui-Young Chung, *Member, IEEE*

***Abstract*—The demand for high-performance computing and rapidly increasing power consumption has increased the necessity for application-specific accelerators. In the datacenter and mobile system, more applications are increasingly relying on accelerators. Field-programmable gate arrays (FPGAs) emerge as a good candidate because they have high programmability and power efficiency. As the number of applications requiring acceleration increases, there is huge demand for FPGAs that support multiple contexts. Previous FPGA designs that support multicontext have various shortcomings such as volatility, poor power efficiency, large performance, area, and reconfiguration overhead. In this article, we propose a spin-transfer torque magnetic RAM (STT-MRAM)-based nonvolatile multicontext FPGA (NVMC-FPGA) that overcomes these shortcomings. We introduce the NVMC-FPGA architecture and operation modes that take advantage of nonvolatility and support multicontext. We also develop the multicontext-aware FPGA computer aided design flow to make the most of the NVMC-FPGA. Compared to the conventional SRAM-based FPGA, when eight identical circuits are mapped, the NVMC-FPGA improves the performance by 15.3% on average and reduces the power consumption by 11.2%–80.7%, depending on the number of simultaneously activated circuits. Moreover, when eight different circuits are mapped, the NVMC-FPGA improves the performance by 58.5% on average and reduces the power consumption by 6.2%–63.3%, depending on the number of simultaneously activated circuits.**

***Index Terms*—Accelerator architectures, design automation, field-programmable gate arrays (FPGAs), multithreading, nonvolatile memory, reconfigurable architectures, spin-transfer torque magnetic RAM (STT-MRAM).**

Manuscript received January 18, 2021; revised May 11, 2021; accepted June 2, 2021. Date of publication June 22, 2021; date of current version April 21, 2022. This work was supported in part by the Research and Development Program for Advanced Integrated-Intelligence for Identification through the National Research Foundation of Korea (NRF) funded by Ministry of Trade, Industry and Energy under Grant 2018M3E3A1057248; and in part by the Ministry of Trade, Industry and Energy (MOTIE) under Grant 10080722 and Korea Semiconductor Research Consortium (KSRC) support program for the development of the future semiconductor device. This article was recommended by Associate Editor W. Zhang. *(Corresponding author: Eui-Young Chung.)*

Jeongbin Kim, Kyungseon Cho, Hongil Yoon, and Eui-Young Chung are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea (e-mail: xtankx123@yonsei.ac.kr; kyungseon.cho@yonsei.ac.kr; hyoon@yonsei.ac.kr; eychung@yonsei.ac.kr).

Yongwoon Song and Hyukjun Lee are with the Department of Computer Science and Engineering, Sogang University, Seoul 04017, South Korea (e-mail: ywsong1985@gmail.com; hyukjunl@sogang.ac.kr).

Digital Object Identifier 10.1109/TCAD.2021.3091440

I. Introduction

As THE Dennard scaling [1] has broken down, power density starts growing, and it limits performance enhancement. Process scaling enables smaller transistors and higher performance, but the increase in power consumption outpaces. Many factors decrease power efficiency. When the processor processes instructions, the energy consumed by computing is just around 10%–40%, and the rest is consumed by memory accesses [2]. Moreover, this additional energy consumption gets larger when the processor is more programmable because high programmability makes data flow from memory to processor more complicated [3]. Processors, such as central processing units (CPUs) and graphics processing units (GPUs) sacrifice the power efficiency for programmability.

Therefore, application-specialized accelerators have been used to obtain power efficiency. Power-sensitive systems, such as the datacenter and mobile already adopt the application specialized accelerator. They contain special-purpose application-specific integrated circuits (ASICs) for image processing, network, video codecs, neural network processing, etc. However, they can process only a single application because ASICs have zero programmability.

Accordingly, field-programmable gate arrays (FPGAs) are emerging for accelerators because they have both programmability and power efficiency. FPGAs are generally used in the datacenter [4]–[6] and mobile [7]–[9] as the accelerator. In terms of application, as deep learning becomes a hot issue recently, FPGAs have become a popular computing platform for a neural network accelerator [10], [11]. As the number of applications that require accelerators increases, FPGAs are integrated into the commercial multithreading computing environment as Intel develops the CPU+FPGA platform [12] for the datacenter. In the multithreading computing environment, application execution accompanies frequent context switching. Therefore, there is high demand for FPGAs supporting multicontext.

However, conventional FPGAs are not adequate to accelerate multiple contexts because conventional FPGAs need reconfiguration for the multicontext operation. The reconfiguration overhead is fatal to performance [13], [14], causing severe performance degradation. Furthermore, they are based on static random-access memory (SRAM) that is a volatile memory. Hence, partial power gating is not viable for inactive contexts when multiple contexts

are mapped. Power gating is critical for efficient power management.

There have been many studies for supporting multicontext more efficiently. Most multicontext FPGA architectures contain a separate configuration memory to store multiple contexts. This technique was first proposed through a dynamically programmable gate array (DPGA) [15]. The DPGA supports multicontext by implementing a multicontext configuration memory outside a look-up table (LUT). Since the DPGA, several multicontext FPGA architectures using the DPGA structure have been proposed [16]–[18]. Moreover, Tabula [19] commercialized the multicontext FPGA with this idea. However, all studies based on the DPGA structure still have reconfiguration overhead when the FPGA performs context switching because the configuration memory and LUT are separated.

To solve this problem, in-LUT multicontext should be implemented. In-LUT multicontext contains multicontext data in the LUT. However, it is hard to implement with SRAM because the cell size of the SRAM is quite large. Thus, a DRAM-based reconfigurable acceleration fabric (DRAF) [20] implements multicontext FPGA by making the LUT data cells using a dynamic random access memory (DRAM) array. As the LUT contains multiple contexts, reconfiguration is not needed when performing context switching. However, the DRAF has several problems caused by characteristics of the DRAM. First, the DRAF has a much lower performance than the conventional SRAM-based FPGA. Second, all circuits must be reconfigured when power is shut down, and the power gating is not possible. Third, its static power is quite large when idle state because it has refresh overhead.

Magnetic random-access memory (MRAM) is a more suitable memory than DRAM and SRAM for implementing multicontext FPGAs. MRAM is nonvolatile memory that contains the data on magnetic tunnel junction (MTJ) cells. It is more suitable for implementing multicontext FPGAs for several reasons. First, the cell size of MRAM is small compared to SRAM [21], so it is easy to implement in-LUT multicontext. Second, nonvolatility makes reconfiguration unnecessary after the power shutdown. Third, partial power gating essential for managing multicontext is viable because MRAM is nonvolatile. Finally, MRAM has a much faster read performance than DRAM, comparable to SRAM. MRAM shows poor write performance, yet it is hidden in FPGAs. When FPGAs are used as accelerates, LUTs perform only memory read operations for reading mapped logic. MRAM write operations in the FPGA only occur when mapping the circuit.

Two types of MRAM are studied for FPGAs: 1) thermal-assisted switching MRAM (TAS-MRAM) and 2) spin-transfer torque MRAM (STT-MRAM). TAS-MRAM is an old-fashioned MRAM and has several drawbacks compared to STT-MRAM. First, its read latency is longer than STT-MRAM [22]. Second, TAS-MRAM requires a large area [23], which becomes a critical issue in implementing multicontext FPGA. Although there is a study that implements in-LUT FPGA supporting two contexts [24], it lacks in experiments and analysis in addition to poor performance and cost of TAS-MRAM-based FPGAs.

In this article, we propose a nonvolatile multicontext FPGA (NVMC-FPGA) based on STT-MRAM. The contributions of this article are summarized as follows.

1) The STT-MRAM-based NVMC-FPGA architecture is presented. It exploits the nonvolatile multicontext LUT (NVMC-LUT) that supports multicontext using multiple MTJ cells and a precharged sense amplifier (PCSA). We demonstrate that the NVMC-FPGA architecture is adequate for multithreading computing environments through runtime context-switching, which can be easily achieved through the PCSA.
2) Management schemes and operation modes for the NVMC-FPGA are proposed. Management schemes include introducing the concepts of the physical context layer (PCL)/logical context layer (LCL) and PCL-free mapping (PFM) technique that improves space efficiency. Operation modes include temporal multicontext (TMC) and spatial multicontext (SMC). TMC supports two types of multithreading: a) coarse-grained and b) fine-grained multithreading. SMC makes multiple circuits operate simultaneously and improves space utilization.
3) Computer-aided design (CAD) flow for the NVMC-FPGA that introduces "context packing" and "MC-placement" stage is proposed. Context packing stage makes SMC possible by mapping multiple circuits to one LCL. MC-placement stage maps multiple circuits more evenly distributed, allowing more circuits to be mapped and incurs less thermal problems.

Experiments are performed in two environments, and the results are compared to the conventional SRAM-based FPGA. When eight identical circuits are mapped, the NVMC-FPGA improves the performance by 15.3% on average and reduces the power consumption by 11.2%–80.7%, depending on the number of simultaneously activated circuits. When eight different circuits are mapped, the performance is improved by 58.5% on average. The reduction of average power consumption is from 6.2% to 63.3%, depending on the number of simultaneously activated circuits. This article is organized as follows. Section II contains the background. Section III explains the details of the NVMC-FPGA. Section IV presents the performance and cost evaluation of the NVMC-FPGA. Finally, Section V draws the conclusion of this article.

II. Background

A. FPGA Architecture

FPGAs provide not only ASIC-like performance but also flexibility because FPGAs program the LUT to implement a logic function. The LUT contains multiple data cells implementing the LUT entries. FPGAs are reconfigurable because data cells can be modified. Conventional FPGAs are classified as an SRAM-based FPGA because they use SRAM for data cells. In the FPGA, a single LUT is packed with flip-flops (F/Fs) for proper management and operation. A LUT with F/Fs

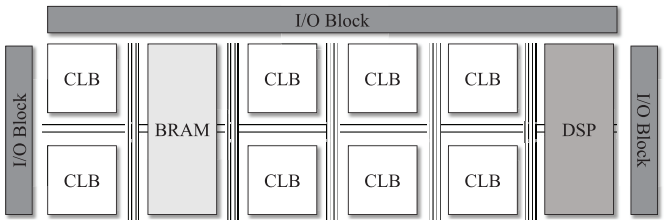

Fig. 1. Island-style FPGA architecture.

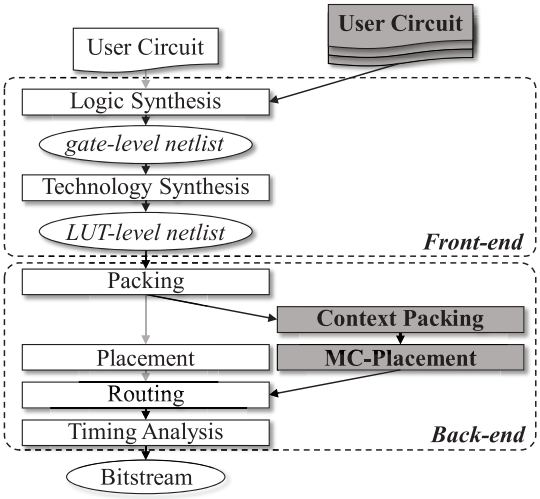

Fig. 2. Conventional FPGA and NVMC-FPGA CAD flow (gray boxes are added and modified for the NVMC-FPGA).

is called a basic logic element (BLE). BLEs are grouped with MUXes and a full-crossbar. This group forms the main logic block of the FPGA, and it is called a configurable logic block (CLB).

The overall FPGA architecture comprises CLBs, block memories (BRAMs) used as a memory device within the FPGA, digital signal processing (DSP) unit for the specific arithmetic operation, and I/O blocks. Most FPGAs mainly adopt an island-style routing architecture [25], as shown in Fig. 1. CLBs are arranged in a grid pattern while BRAMs and DSPs are interlaced. I/O blocks wrap around them to communicate with the outside. Between elements, there are interconnects to communicate with each element. The interconnect includes connection and switch boxes. Connection boxes connect wires and each element. Switch boxes are located at the intersection and connect the intersecting wires. These boxes commonly consist of MUXes, and these MUXes select the input and output signals using the attached memory cells. FPGAs reconfigure interconnects by changing data of memory cells that are attached at interconnect MUXes.

B. FPGA CAD Flow

The FPGA CAD flow is a sequence of processes that synthesizes the user-input circuit and maps it to the FPGA. White boxes in Fig. 2 represent a general FPGA CAD flow. When the circuit hardware description language (HDL) is given to the FPGA CAD flow, it generates a bitstream that maps the circuit to the FPGA. Compared to an ASIC CAD flow, "technology

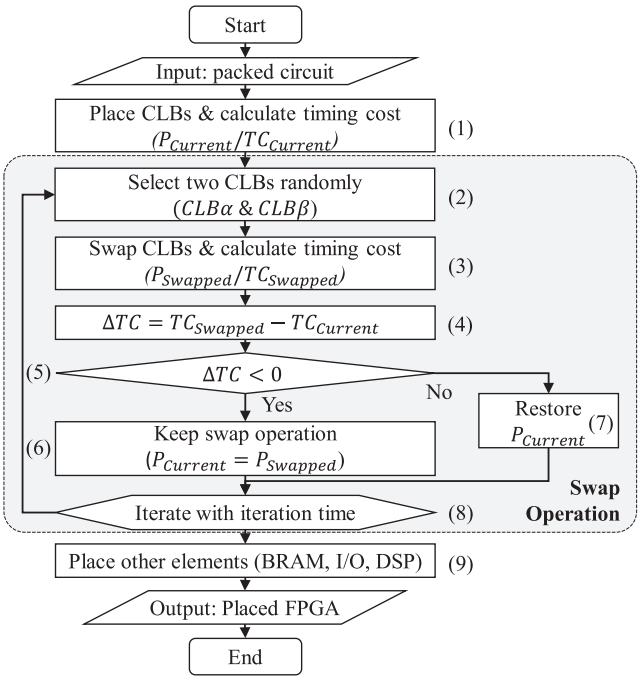

Fig. 3. Conventional placement sequence.

synthesis" and "packing" are added in the FPGA CAD flow. Technology synthesis generates an LUT-level netlist from a gate-level netlist, and packing groups LUT-level netlist elements to FPGA hardware elements and determines the FPGA size by considering hardware resources of the context. For the NVMC-FPGA, the conventional FPGA CAD flow is modified, as shown in Fig. 2 with gray boxes. Context packing is added to the conventional FPGA CAD flow, and MC-placement is optimized "placement" for the multicontext environment.

The main purpose of the placement is to place FPGA hardware elements to FPGA devices with minimizing a critical path delay that is a delay of the longest path. Since the FPGA size is already determined through packing, the conventional placement places the hardware resources based on timing optimization under the area constraint. Fig. 3 represents the conventional placement sequence. It is based on the timing-driven placement [26], which is to minimize the critical path delay by using a swap operation. The swap operation swaps places of each CLB. It minimizes the critical path delay by putting a condition (5).

First, it receives a packed circuit from packing, containing the list of FPGA hardware elements. Subsequent steps of the conventional placement sequence is as follows. Each sequence matches the number in Fig. 3.

1) It places all CLBs randomly and calculates the timing cost of this placement. $P_{Current}$ is a current placement after placing CLBs randomly, and $TC_{Current}$ is its timing cost. The timing cost is the sum of the delay of all connections in the current placement, considering each connection's criticality. After that, it enters the swap operation.
2) The first step of the swap operation selects two CLBs (CLBα, CLBβ) randomly from the current placement.

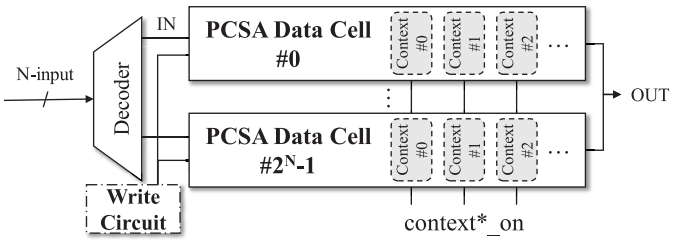

Fig. 4. Proposed NVMC-LUT architecture.

CLBα can be an empty CLB. If an empty CLB is selected, it just moves one CLB to the empty CLB place.

3) It swaps the selected CLBs and calculates the timing cost of this placement. P_{Swapped} is a placement after swapping, and $\text{TC}_{\text{Swapped}}$ is its timing cost.
4) Calculate ΔTC, which is the timing cost change when swapping CLBs.
5) If ΔTC is less than 0, it means that the swap operation decreases the timing cost.
6) Hence, if the condition in (5) is met, it keeps performing swap operations. It selects P_{Swapped} over P_{Current}.
7) If the condition in (5) is not met, it restores P_{Current}. It means that it does not perform the swap operation.
8) It iterates from step (2) to (7) as many times as the number of CLBs in the circuit.
9) Finally, it places other elements (BRAM, I/O block, DSP) and outputs placed FPGA.

This sequence is simplified in this description for easier understanding. We just explain the parts that are necessary to understand our proposed method. The cost of wiring and calculating the timing cost is excluded in this description. The detail of this algorithm is explained in [26]. Consequently, the placement places FPGA hardware elements (especially, CLBs) while minimizing the critical path delay through the above sequence.

III. NVMC-FPGA : STT-MRAM-Based Nonvolatile Multi Context FPGA

In this section, we explain the detail of our proposed NVMC-FPGA. It is divided into three parts. First, NVMC-LUT based on STT-MRAM, which becomes a major building block for the NVMC-FPGA. Second, the architecture and operation modes of the NVMC-FPGA. Finally, the CAD flow for the NVMC-FPGA supports multicontext FPGA architecture and utilizes the NVMC-FPGA's characteristics.

A. STT-MRAM-Based Nonvolatile Multi Context Look-Up Table

Previous research on STT-MRAM-based LUTs is limited to supporting a single context and reveals other severe limitations. Since LUT operations in [23] must be synchronized to the clock signal, it has a huge limitation when configuring the FPGA with the combinational logic. In [27], LUT operations are not synchronized to clocks. However, it has a configuration limitation because the LUT only operates normally according to the clock level. Although the method in [28] eliminates the above limitations, it consumes very high static power.

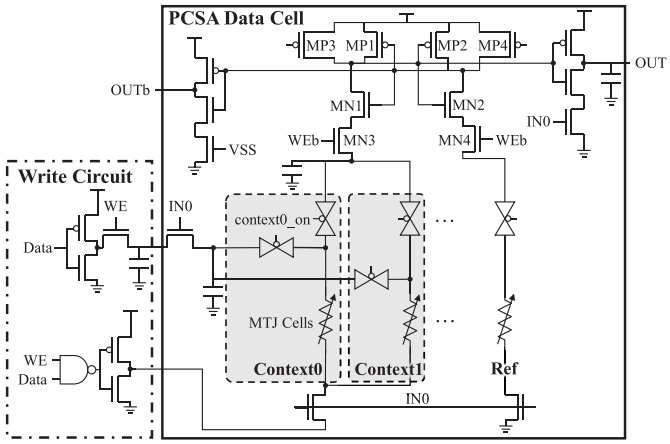

Fig. 5. Schematic diagram of the one PCSA data cell and the write circuit in the proposed NVMC-LUT.

The STT-MRAM-based NVMC-LUT [29] solves these problems by employing the PCSA [30] as a sense amplifier. The PCSA has a characteristic of high speed and high reliability, so it is more efficient for the MTJ cell. Furthermore, the context switching can be operated during runtime because the sensing latency of the PCSA is extremely small (about 200 ps [30]). From our experiment results, it is only 0.02% of the critical path delay on average. Therefore, the context-switching operation has a little effect on the clock frequency.

Fig. 4 represents the NVMC-LUT architecture. N-input NVMC-LUT consists of the following elements. The decoder to select one data cell through an input signal, 2^N PCSA data cells for storing all data of N-input LUT, and a write circuitry that writes data to the PCSA data cell. Each PCSA data cell comprises the PCSA and contains all context data for each bit address. When an N-bit signal is input to the LUT, the decoder selects one of the data cells and activates it (If bit address "000001" is the input, data cell #1 is selected). After activating one of the data cells, the context is selected by "context*_on" signals. As a result, the NVMC-LUT outputs the data corresponding to the selected context and address.

Fig. 5 represents the schematic diagram of the PCSA data cell and write circuit. The PCSA data cell is divided into two parts by sense amplifier transistors (MN3, MN4). The upper part is the sense amplifier and the lower part is the memory part that contains MTJ data cells. The memory part of NVMC-LUT contains multicontext MTJ cells, and they are selected by context*_on signals.

When the PCSA data cell is selected, the "IN0" signal toggled. The data cell part is connected to a write circuit or a sense amplifier according to the write enable (WE) signal and performs a read or write operation. The details of the read operation are as follows. As the IN0 signal toggled, the latches (MP1, MP2, MN1, MN2) are precharged through the precharge transistors (MP3, MP4). Precharged latches read the data from the MTJ cell and hold the data, and it is connected to the output node through the output inverter on the right side. In the writing operation, the memory part is connected to the write circuit when the "WE" signal is toggled to high, and the data is written to the MTJ cell through the write circuit.

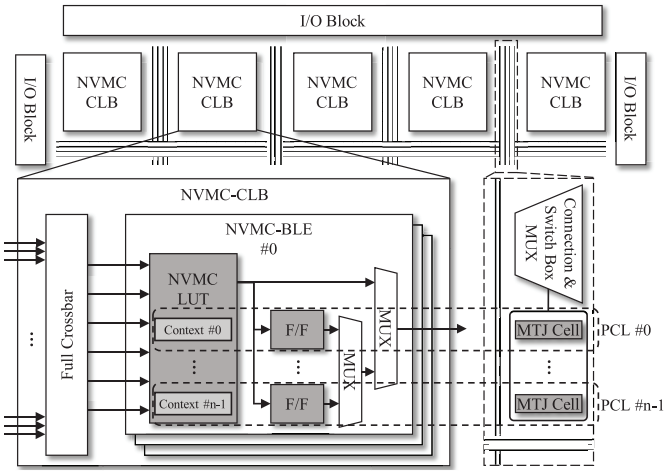

Fig. 6. NVMC-FPGA architecture.

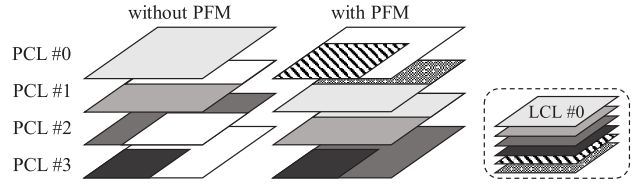

Fig. 7. PCLs and LCLs when mapping multiple contexts with or without PFM mode (same color represents the same LCL, and white color represents empty space).

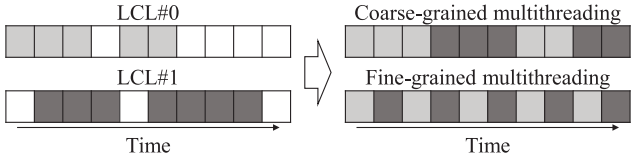

Fig. 8. Fine-grained and coarse-grained multithreading in the NVMC-FPGA.

B. Architecture and Operation of the NVMC-FPGA

We propose the NVMC-FPGA that exploits the NVMC-LUT. To support nonvolatility and multicontext, several parts of the architecture are changed compared to the conventional FPGA. Besides, we propose various operation modes suitable for this architecture to increase the efficiency of the NVMC-FPGA.

1) Architecture of the NVMC-FPGA: Fig. 6 represents the architecture of the proposed NVMC-FPGA. The gray boxes represent the changes from the conventional FPGA. In the logic part, conventional LUTs in each CLBs are all replaced with NVMC-LUTs and as many F/Fs as the number of contexts. In the interconnection part, the single SRAM cell at interconnect MUXes is replaced with as many MTJ-cells as the number of contexts. With these changes, the NVMC-FPGA is architecturally able to support multicontext and has nonvolatility.

2) Operation for the NVMC-FPGA: To describe the operations of the NVMC-FPGA, we first define two terms: 1) PCL and 2) LCL.

PCL is a partition of the NVMC-FPGA based on the physical location of FPGA elements. As shown in Fig. 6, one F/F and one context data in NVMC-LUTs at each BLEs, and a single MTJ-cell at each interconnect MUXes are grouped and form one PCL. LCL is a partition of the NVMC-FPGA that is logically connected and can operate simultaneously in the same time domain. To simplify the FPGA design, we can make LCL to be same as PCL. However, in this case, as many LCLs as the number of PCLs can only be mapped regardless of the remaining space in PCL.

To solve this problem, the NVMC-FPGA supports the PFM operation mode. PFM is a mapping technique that allows the LCL to be mapped to multiple PCLs, not limited to a single PCL. It makes the NVMC-FPGA map multiple contexts more flexibly and can map additional contexts to the remaining space regardless of PCL. Fig. 7 shows PCLs and LCLs of the NVMC-FPGA when mapping multiple contexts with or without the PFM mode. Different colors represent different LCLs of various sizes, and four planes represent four PCLs. When mapping multiple contexts without the PFM mode, each PCLs can map only a single LCL regardless of the remaining space, and it is shown on the left side of Fig. 7.

When mapping multiple contexts with PFM mode, each PCL can map multiple LCLs, allowing an LCL to be mapped to the remaining space of multiple PCLs. The right side of Fig. 7 represents PCLs and LCLs of the NVMC-FPGA when mapping multiple contexts with the PFM mode. In this example, two more LCLs (patterned color) can be mapped than the NVMC-FPGA without the PFM mode. Thus, in a real environment, where circuits of various sizes are mapped, PFM improves space efficiency. To map multiple contexts with the PFM mode, hardware elements of each PCL should know which LCL is mapped. The implementation of PFM will be explained after explaining the multithreading methods because the possible multithreading method varies depending on how it is implemented. Next, we discuss how PCL and LCL are used to map multiple contexts.

TMC: Each LCL can be switched in runtime—we call this *TMC*. It is possible because the proposed NVMC-LUT architecture allows switching the context with little timing overhead by employing the PCSA. TMC supports coarse-grained and fine-grained multithreading. These terms are generally used in computing systems, and we use them to depict the multithreading behavior of the NVMC-FPGA.

Fig. 8 shows an example of the coarse-grained and fine-grained multithreading in the NVMC-FPGA. Each colored block means a single clock operation, and both LCLs are assumed to have the same clock frequency. The left side represents a sequence of operations in each clock cycle for two LCLs without multithreading. The right side represents a sequence of operations when employing two multithreading techniques. The operation and hardware resource usage of each multithreading are as follows.

If the NVMC-FPGA runs with coarse-grained multithreading, it performs a context switching after a sequence of operations ends. It shows little performance loss because context switching occurs infrequently. The context switching occurs as follows. The NVMC-LUT performs the context switching by changing the context*_on signal. In terms of

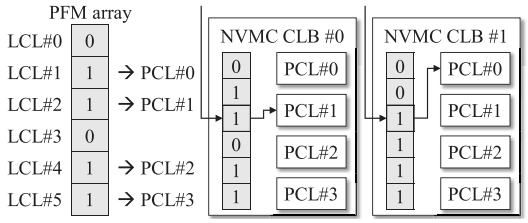

Fig. 9. PFM array.

F/F, only one F/F is needed for each BLE. This would require flushing pipelines upon context switching, yet it reduces the area overhead of F/Fs. In terms of BRAM, a single LCL can use the entire BRAM space because data usage of BRAM is not continuous for each sequence of operations.

When the NVMC-FPGA runs with fine-grained multithreading, it performs context switching after a single clock operation ends. It guarantees fairness among contexts. However, frequent context-switching can lead to performance loss due to its overhead. Unlike coarse-grained multithreading, all of LCL's intermediate results should be maintained when context switching occurs because context switching occurs every cycle. For this, there should be as many F/Fs as the number of PCLs for each BLE. Furthermore, if multiple LCLs use the BRAM, they should share the BRAM space because each LCL's data should be maintained even if context-switching occurs.

SMC: Multiple circuits can also be mapped in a single LCL—we call this *SMC*. SMC has several advantages. First, multicontext can be implemented in the conventional FPGA with a single LCL. Second, circuits mapped with SMC can operate simultaneously. Third, if the NVMC-FPGA uses SMC, it can save power consumption by turning off the unused circuits because the NVMC-FPGA supports partial power gating. Furthermore, SMC increases the utilization of FPGA. When mapping small circuits to the FPGA, another circuit can be mapped to the remaining space. SMC is supported through the context packing stage of the NVMC-FPGA CAD tool flow.

3) Implementation of PFM: To map contexts with PFM mode, the hardware elements of each PCL should know which LCL is mapped. There are two ways to achieve this.

The first implementation is that the context mapping information is stored in the external memory and is provided externally without any additional hardware element. When a particular LCL becomes active, it selects the corresponding hardware resources from multiple PCLs. This implementation does not have additional hardware overhead. However, if PFM is implemented in this way, the NVMC-FPGA in PFM mode cannot operate with fine-grained multithreading because it is hard to perform context switching quickly.

The second implementation is adding an additional hardware element called a "PFM array" for quick context-switching. Fig. 9 represents the architecture and operation of the PFM array. The PFM array is an STT-MRAM array to store information about elementwise LCL-PCL mapping. It contains n bit data, and each bit represents a single LCL. The value "1" in the array indicates which LCL is mapped to each PCLs in ascending order. First "1" is mapped to PCL#0, and next "1" is mapped to PCL#1. It means LCL#1 & #2 are mapped to PCL#0 & #1. In the right example, LCL#2 chooses PCL#1 and PCL#0 LUT in CLB#0 and CLB#1, respectively. Therefore, the NVMC-FPGA in the PFM mode can operate with fine-grained multithreading through PFM array. However, the hardware overhead of the PFM array is nonnegligible because all interconnect MUXes and CLBs should have these arrays.

In this article, we implement PFM without the PFM array because it is best in the area and does not worsen overall performance. In the case of area, this architecture can perform PFM without additional hardware "PFM array." Therefore, this implementation makes multiple contexts can be mapped most efficiently with the same area. In the case of the overall performance, operating only with coarse-grained multithreading does not worsen overall performance. Furthermore, when fine-grained multithreading is essential, the NVMC-FPGA without the PFM array can also operate with fine-grained multithreading if multiple contexts are mapped without the PFM mode.

C. CAD Flow for the NVMC-FPGA

We modify a CAD flow for the NVMC-FPGA to support the following functions. First, it supports multicontext in order that multiple circuits can be mapped to the NVMC-FPGA structure. Second, SMC is supported by integrating multiple circuits into a single LCL. Finally, partial power gating is provided to reduce the power consumption of the FPGA. We implement these functions with verilog-to-routing (VTR) tool [31]. The VTR tool is the most popular opensource FPGA CAD tool and is broadly used in academia.

The CAD flow for the NVMC-FPGA is shown in Fig. 2. Three parts are modified from the conventional FPGA CAD flow to support the above functions. First, all CAD flow steps are modified to support multicontext functions. Each step of the CAD flow operates for multiple circuits and stores the output from all circuits. Second, the context packing is added to support SMC. Integrated circuits can be operated simultaneously and marked distinctly to support partial power gating. Finally, the conventional placement is modified for a multicontext-aware placement (MC-placement). When multiple circuits are mapped, it allows each circuit to be mapped dispersedly to the entire FPGA structure. The detailed process of the context packing and the MC-placement is as follows.

1) Context Packing: The context packing is added between the packing and placement to support SMC. The context packing packs multiple circuits to a single LCL. Processes of context packing are as follows. First, we pack elements of each circuit into a single circuit. That enables several circuits to be mapped to a single LCL. The FPGA size is redetermined by considering the number of hardware resources in the packed context. Next, CLBs, multipliers, and interconnect MUXes are labeled with its circuit's name (i.e., context number), allowing partial power gating. Therefore, power gating can be performed on CLBs, multipliers, and interconnects.

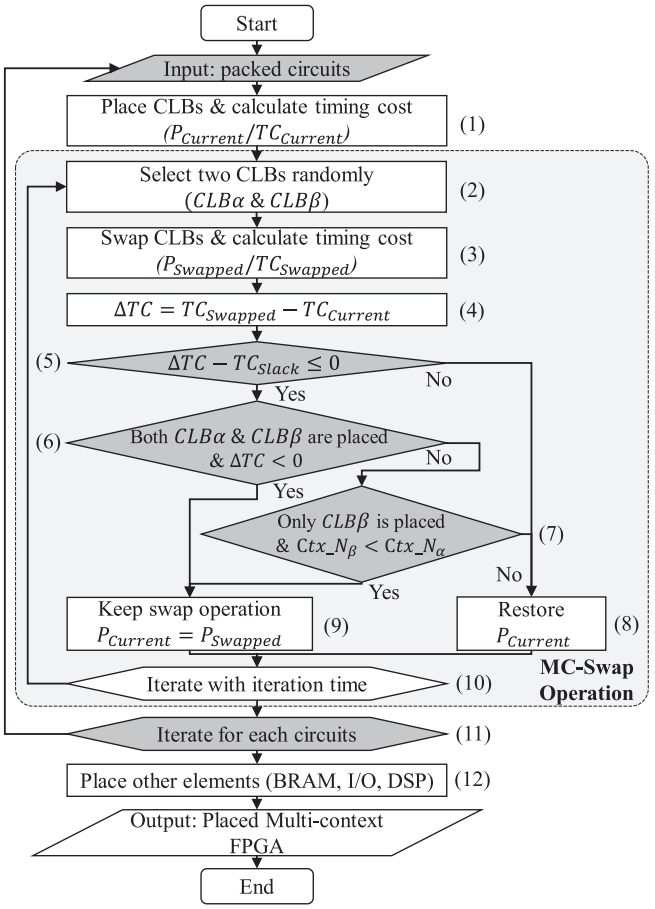

Fig. 10. MC-placement sequence.

BRAMs are excluded in power gating because it is based on SRAM. Consequently, the context packing packs multiple circuits through the processes above and gives the list of circuits to the MC-placement.

2) MC-Placement: When a user wants to map multiple circuits on the NVMC-FPGA, each circuit can be placed on each PCLs by repeating the conventional placement. However, it causes the circuits to be unevenly mapped to only a few CLBs. Therefore, we propose the MC-placement that maps multiple circuits more evenly distributed. There are two advantages to MC-placement. First, it allows more circuits to be mapped more evenly. Because the NVMC-FPGA supports PFM, more circuits than the number of PCLs can be mapped. Second, it solves a thermal issue. Circuits mapped only on a few CLBs causes thermal issues. The thermal issue worsens when the FPGA performs fine-grained multithreading, potentially increasing the error rate [32].

Fig. 10 shows the MC-placement sequence. Gray boxes are changed from the conventional placement. The beginning of the placement is modified to receive multiple circuits. The condition in (7) is the main algorithm of MC-placement. It is added to map the circuit more evenly. Finally, the iterate point (11) is added to map multiple circuits in the NVMC-FPGA.

The details of MC-placement are as follows. First, step (1) performs the placement of one circuit. Steps (2)–(4) swap two selected CLBs and calculate the timing cost of swapped placement. Step (5) checks if the timing cost increases, which is the same as the conventional placement (TC_{Slack} will be explained later). If the condition in (6) is met, this swap does not affect the distribution of context and only improves the critical path delay. Therefore, the swap operation is performed to optimize the critical path delay. If the condition in (6) is not met, we determine whether the swap operation distributes the context more evenly. MC-placement checks this with the condition in (7). Ctx_N_α represents the number of contexts that are mapped on CLBα. When each circuit finishes from steps (1) to (11), *Ctx_N*s of all CLBs are determined. The condition in (7) consists of two subconditions. The former checks if only CLB_β is placed. If it is true, the current swap operation is actually a move operation, not a swap operation. The latter checks if Ctx_N_β is less than Ctx_N_α. Satisfying both conditions indicates that the swap is beneficial because it distributes contexts on two CLBs more evenly. Consequently, the condition in (7) makes the swap operation move CLB_β to a less occupied CLBα.

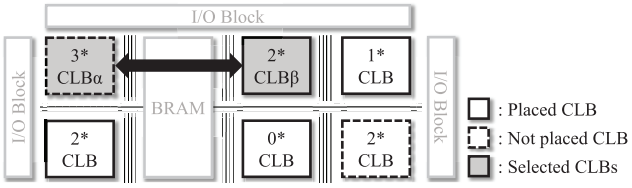

Fig. 11. Example of the MC-swap operation.

Fig. 11 shows an example of the MC-swap operation to explain the condition in (7). Terms "placed CLB" and "not placed CLB" are determined with respect to the circuit currently performing placement operation. In this example, three circuits are already placed, and the fourth circuit is being placed. The asterisk numbers on CLBs are *Ctx_N* of each CLB after three circuits are placed by iterating from (1) to (11) three times. By step (2), CLBs are selected to operate a swap of two CLB's in gray boxes of Fig. 11. In this example, it is assumed that the condition in (5) is met, and the condition in (6) is not met because only CLB_β is placed. CLB_α and CLB_β in Fig. 11 are at the swapped position. The former subcondition in (7) is met because CLBβ is only placed. The latter subcondition in (7) is met because Ctx_N_β is less than Ctx_N_α, and thus it keeps performing the swap operation. As a result, Ctx_N_α and Ctx_N_β become 3 and 3, and contexts are evenly placed.

The MC-Swap operation only performs the swap operation when both conditions (5) and (7) are met. Thus, the MC-placement can optimize both the critical path delay and context distribution. Besides, we add the timing cost slack (TC_{Slack}) to condition in (5) to favor even distribution of contexts over slight increasing timing cost. The condition in (5) is met even if $TC_{Swapped}$ is larger than $TC_{Current}$ due to adding TC_{Slack}. Thus, more swap operations can reach step (7).

After a current circuit is placed, we update *Ctx_N* for all CLB. Steps from (1) to (11) are repeated for each circuit. We place other FPGA hardware elements such as BRAM, I/O block, DSP. Finally, the MC-placement produces the placed multicontext FPGA that optimizes both context distribution

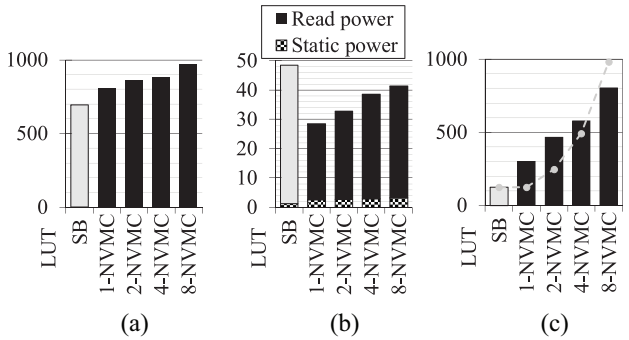

Fig. 12. Simulation result for the NVMC-LUT. (a) Read latency (ps). (b) Power (μW). (c) Area (μm^2).

and critical path delay under the area constraint given from context packing.

IV. Evaluation

This section shows the performance and cost of the NVMC-FPGA. First, we show the simulation results of NVMC-LUT. Then, we measure the performance and cost of the NVMC-FPGA with the VTR tool [31] and compare them with the conventional SRAM-based FPGA, including the analysis on the effect of our CAD flow.

A. NVMC-LUT

1) Experimental Setup: The size of the LUT refers to the number of inputs connected to the LUT, and it affects the delay and area of circuits mapped on the FPGA. We choose the size of the LUT to be 6. It is the optimal point of the delay and area [33], which is most commonly used in the industry. A PSPICE simulation is used to evaluate the performance of the NVMC-LUT. Parameters are chosen to produce a yield close to 100%. The yield is derived by running the Monte-Carlo simulation 1000 times. NCSU 45-nm process technology model parameters and the small macro model of the MTJ are used for all simulations. The tunnel magnetoresistance ratio (TMR) is 200%.

For the NVMC-FPGA experiments, we derive parameters of the 6-input NVMC-LUT, including the read latency, power consumption, and area. We compare these results with an SRAM-based 6-input LUT (SB-LUT). The architecture of the SB-LUT is based on the SRAM LUT architecture in [25], and the architecture parameters are obtained from this article [33]. All results of SB-LUT are obtained from the PSPICE simulation.

2) Experimental Result: Fig. 12 shows the read latency, power consumption, and area for the NVMC-LUT compared to the SB-LUT. The numbers in front of the NVMC on the x-axis represent the number of maximum contexts that can be mapped to each NVMC-LUT.

Fig. 12(a) shows the read latency of the NVMC-LUT. As the number of contexts increases, the sense amplifier's size also increases, which leads to a slight increase in the read latency. As a result, the read latency increases by 15.5%–39.5% compared to the SB-LUT. However, the FPGA performance is mostly affected by the interconnect delay and the area. Thus, the increase of LUT read latency does not significantly affect the FPGA performance.

Fig. 12(b) shows the power consumption of the NVMC-LUT. The read power is consumed when the LUT's output is switched, while the static power is always consumed regardless of output data transition. The static power of NVMC-LUT is almost doubled than SB-LUT's when it has one context, and it is due to the precharge operation of the PCSA in NVMC-LUT. Therefore, the power consumption of LUTs depends on the transition rate of the output data.

Fig. 12(c) shows the area of the NVMC-LUT. The gray line represents the used SB-LUT area when multicontexts are mapped. In the multicontext environment, the FPGA that uses SB-LUT maps multiple contexts in multiple LUTs separately. Therefore, it is estimated simply by multiplying the area of the single context SB-LUT by the number of contexts to compare with the NVMC-LUT's area. Even though the SRAM's cell size is larger than STT-MRAM, the area of the NVMC-LUT is about 2.5 times larger than the area of the SB-LUT when the number of the context is 1. It is because the NVMC-LUT implements a large sense-amplifier (PCSA) for the performance. However, the area increase can be hidden from the perspective of the FPGA. LUTs take only a part of the entire FPGA chip, whereas the interconnects take a larger area. In addition, as the number of contexts increases, the area difference significantly decreases. When the number of contexts is 8, the NVMC-LUT takes less area than the SB-LUT. It means the NVMC-LUT is efficient in the multicontext environment that supports a large number of contexts. The detailed experimental results of the FPGA area are presented in Fig. 18.

B. CAD Flow for the NVMC-FPGA

1) Experimental Setup: We discuss the setup in terms of FPGA models for the NVMC-FPGA, various workloads, and CAD tool environments. These setups correspond to all NVMC-FPGA experiments. The details of these are as follows.

FPGA Models: FPGA architecture models, including a baseline FPGA, are based on Intel Stratix IV GX (EP4SGX230) [34], which is provided by the VPR CAD tool [35]. We modify LUT parameters with the values from Fig. 12. The baseline FPGA architecture is a conventional SRAM-based FPGA with a single PCL. We choose the NVMC-FPGA with 8 PCLs, the maximum number of PCL that the NVMC-FPGA supports. The architecture is shown in Fig. 6 ($n = 8$).

Workloads: Workloads are represented in Table I. Four benchmark suites are used: 1) Machsuite [36] benchmark; 2) Vivado high-level synthesis (HLS) design examples [37]; 3) verilog circuits in VTR[31]; and 4) in-house neural network applications. Machsuite is a benchmark suite that evaluates a hardware accelerator and includes various workloads. In-house neural networks contain two neural network types: 1) binary neural network [38] and 2) softmax function [39]. In Table I, the bold number on each benchmark represents the hardware element that affects the FPGA size most. It indicates that when

TABLE I
BENCHMARK LIST

Index	Benchmark	Source	Description	I/O Block	LUT	F/F	DSP
aes	aes	Machsuite	AES encryption	**193**	1280	474	0
bfb	bfs_bulk	Machsuite	Breadth-first search	**389**	812	380	0
bn4	BNN4_xnor	In-house	Neural Network	**654**	141	2	0
cor	cf_cordic_v_18_18_18	VTR	Image Processor	111	**4353**	2052	0
ctz	__builtin_ctz	Vivado HLS	CTZ encoder	**46**	73	43	0
dif	diffeq	VTR	Differential	**258**	595	193	5
fft	cf_fft_256_8	VTR	Fast Fourier Transform	69	**6991**	3147	0
kmp	kmp	Machsuite	String matching	**208**	536	264	0
nw	nw	Machsuite	DNA alignment	**269**	1222	498	0
pm	mkPktMerge	VTR	Packet Processing	467	451	60	**459**
s3d	stencil_stencil3d	Machsuite	Stencil computation	230	**3093**	504	0
sm8	Softmax_dot8	In-house	Neural Network	**1168**	11158	264	0
sr	sort_radix	Machsuite	Sorting	**387**	1168	311	0
vit	viterbi_decoder	Vivado HLS	Viterbi decoder	13	**24599**	17234	0

the multiple circuits are mapped, each workload's FPGA size is bounded by the bold hardware element.

CAD Tool: To evaluate the NVMC-FPGA, VTR [31] is used. As mentioned in Section III-C, VTR is the most popular opensource FPGA CAD tool. As the name implies, it takes a circuit in verilog as an input and delivers the final FPGA mapping that has completed routing. The CAD flow sequence of VTR is the same as Fig. 2, and VTR consists of several tools; ODIN-II [40] for the logic synthesis, ABC [41] for the technology synthesis, and VPR [35] for the entire backend sequence. VPR allows power estimation through SPICE CMOS technology information. In this article, it is estimated based on the 45-nm predictive technology model [42] obtained from the nano-CMOS tool [43], and both static and dynamic power are considered. VTR takes verilog code as an input, but Machsuite and Vivado HLS design examples are written in C/C++ languages. Therefore, the Vivado HLS Tool [37] is used for converting from C/C++ to verilog code. We use Yosys [44] for the logic synthesis instead of "ODIN-II" as it offers better optimization and proper multimodule support. These CAD tools are used for both the baseline and NVMC-FPGA for a fair comparison.

2) Improvement by MC-Placement: In this article, we propose the MC-placement to distribute contexts more evenly in the multiple PCLs environment. We compare the context distribution of the NVMC-FPGA using the conventional placement and the MC-placement. The conventional placement does not support multicontext. Therefore, in this experiment, input and circuit iteration parts are added to the conventional placement for supporting multicontext. However, the main algorithm of the conventional placement is maintained, and the conventional placement does not consider context distribution at all. We use the standard deviation to show the context distribution, measured with the number of contexts mapped to each CLB (*Ctx_N*). The standard deviation is the measure of distribution, and low standard distribution means that contexts are more evenly distributed.

Fig. 13 shows a distribution of contexts after mapping the workload "diffeq" using conventional and MC-placement.

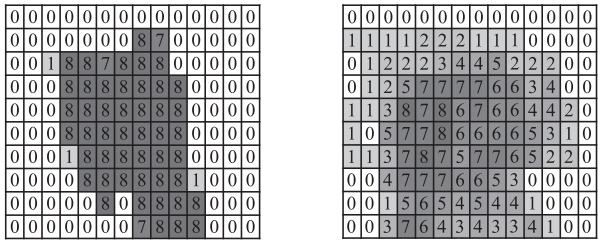

Fig. 13. Number of contexts placed in each CLB for workload diffeq mapped using (a) conventional placement and (b) MC-placement.

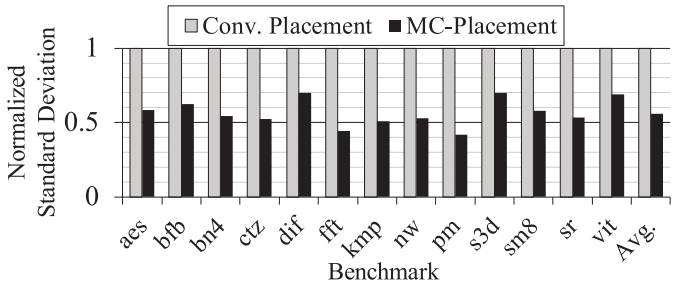

Fig. 14. Context distribution for conventional and MC-placement (normalized to the conventional placement).

The number in boxes represents the number of contexts mapped to each CLB in the FPGA. In Fig. 13(a), contexts are concentrated in a few CLBs because the conventional placement does not consider the multicontext mapping. However, when the proposed MC-placement is used, contexts are more evenly distributed without increasing the critical path delay. This distribution is shown in Fig. 13(b). In terms of standard deviation, Fig. 13(b) shows less standard deviation than Fig. 13(a) as most values in Fig. 13(b) are close to the average. It indicates that the proposed MC-placement can map contexts more evenly distributed.

Fig. 14 compares the standard deviations of the NVMC-FPGA that maps 8 contexts with the conventional placement and the proposed MC-placement. "Cor" workload is excluded because it is fully crowded. The NVMC-FPGA using

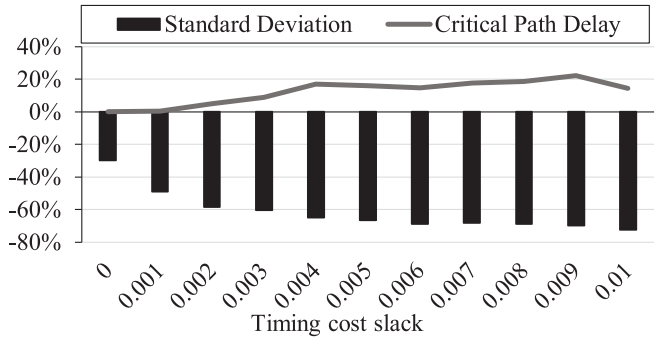

Fig. 15. Context distribution with respect to timing cost slack for stencil3d (normalized to the conventional placement).

TABLE II
IMPROVEMENT OF CONTEXT DISTRIBUTION COMPARED TO THE TIMING COST SLACK OF 0 WHEN ALLOWING THE 5% INCREASE IN THE CRITICAL PATH DELAY

Workloads	Improvement of the standard dev. from the timing cost slack of 0 (% point)	Timing cost slack
aes	-25.1%	0.006
bfb	-16.0%	0.007
bn4	-17.3%	0.002
dif	-32.1%	0.009
fft	-1.4%	0.0001
kmp	-6.6%	0.01
nw	-21.0%	0.009
pm	-8.4%	0.009
s3d	-28.6%	0.002
sm8	-5.2%	0.0001
sr	-17.4%	0.008
vit	-27.1%	0.0001
Average	-17.2%	

MC-placement shows a 44.1% less standard deviation than the conventional placement. It successfully allows more circuits to be mapped and incurs fewer thermal problems.

Moreover, Fig. 15 shows the effect of the timing cost slack on the critical path delay and context distribution when "stencil3d" is mapped. As mentioned above, the timing cost slack is the parameter that allows performing swap operation even if the critical path delay increases. It makes contexts more evenly distributed with a sacrifice of the critical path delay. In these experiments, we sweep the timing cost slack from 0 to 0.01 to find the optimal point. When the timing cost slack is 0.002, the standard deviation decreases 28.6% more than the case of 0 slack, while the critical path delay only increases by 5%.

Table II represents the improvement of the context distribution for varying the timing cost slack. When we allow the 5% increase in the critical path delay, the standard deviation decreases by 1.4% point (fft), to 32.1% point (dif), and 17.2% point on average. "ctz" is excluded because there is no delay and context distribution change with varying the timing cost slack. It is due to the small FPGA size, which leads to little mapping change. Consequently, the timing cost slack makes the circuit more distributed with little critical path delay increasing.

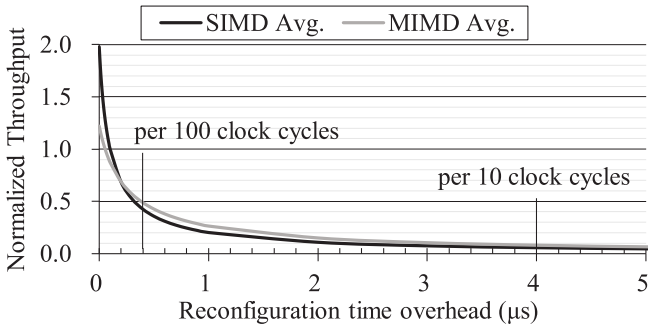

Fig. 16. Normalized throughput for the SRAM-based FPGA with dynamic reconfiguration (normalized to the NVMC-FPGA).

C. Performance and Cost of the NVMC-FPGA

This section shows the performance and cost of the design using the proposed NVMC-FPGA. The DRAF [20] is excluded from the comparison because it is a volatile FPGA and has three times slower performance than the baseline. Because the baseline has a single PCL and does not support TMC structurally, we implement multicontext on the baseline in two ways.

First, the baseline performs multicontext through dynamic reconfiguration. In this case, we show the experimental result with dynamic reconfiguration when the baseline has the same area as the NVMC-FPGA. For each workload, we determine the area occupied by the NVMC-FPGA with eight contexts, and we use the baseline FPGA with the same size to pack as many contexts as possible. Thus, both FPGAs are fully utilized. Dynamic reconfiguration time overhead depends on various factors, such as the FPGA controller and the circuit mapped. We borrow the shortest reconfiguration time (4 μs) in [13] as dynamic reconfiguration time overhead for a fair comparison.

Second, the baseline performs multicontext through SMC. Unlike reconfiguration experiments, this experiment proceeds with the same number of contexts for both FPGAs. Therefore, the area of FPGA for each workload is different, and the area of the baseline FPGA and NVMC-FPGA is also different. Eight contexts are mapped on both the baseline and the NVMC-FPGA, and all contexts in the baseline are mapped into a single PCL and operate with SMC. The NVMC-FPGA maps each circuit separately to each PCL via the MC-placement. Therefore, both timing and context distribution are optimized for the NVMC-FPGA.

For each implementation, we first compare the performance and cost between the baseline and the NVMC-FPGA when the same eight circuits are mapped. It is called single instruction multiple data (SIMD) experiment. Next, we compare the performance and cost when two different circuits are mapped, and it is called the multiple instruction multiple data (MIMD) experiment.

1) Versus Baseline With Dynamic Reconfiguration: Fig. 16 represents the experimental results of the normalized throughput for the baseline with dynamic reconfiguration. They are normalized to the NVMC-FPGA and conducted by sweeping

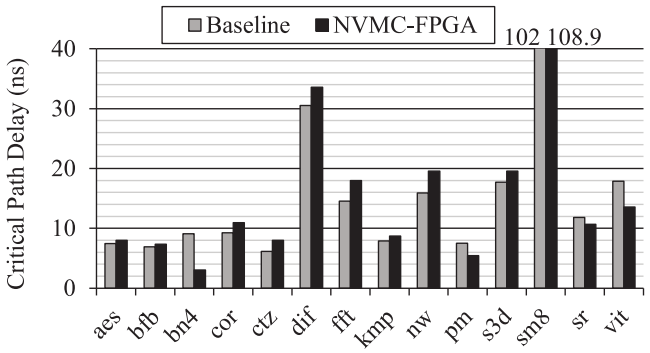

Fig. 17. Critical path delay for the baseline and NVMC-FPGA (SIMD).

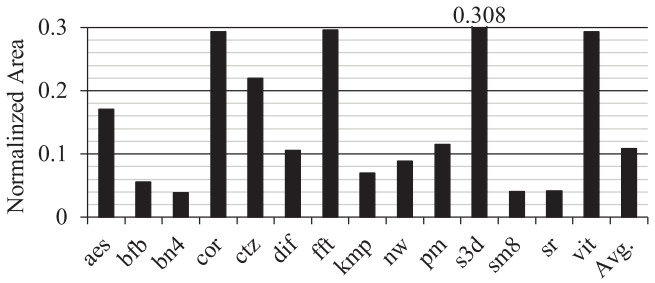

Fig. 18. Normalized area for the NVMC-FPGA (SIMD).

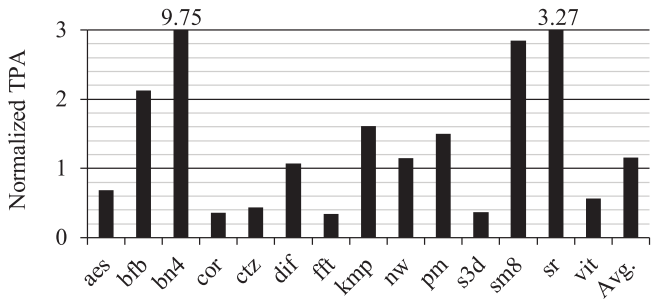

Fig. 19. Normalized TPA for the NVMC-FPGA (SIMD).

the reconfiguration time overhead of the baseline. It is assumed that the dynamic reconfiguration is performed every 10 or 100 clock cycles (coarse-grained multithreading). Both SIMD and MIMD experimental results are derived through performing all possible cases and averaging these results (14 and 91 cases in SIMD and MIMD, respectively).

As shown in Fig. 16, the reconfiguration time overhead is quite critical to the FPGA performance. When the reconfiguration time overhead is 4 μs per 10 and 100 clock cycles (from [13]), the baseline's throughput is only 5.6%/8.1% (SIMD/MIMD) and 42.4%/48.7% (SIMD/MIMD) of the NVMC-FPGA's throughput. As a result, the baseline that performs multicontext through dynamic reconfiguration has limitations in a multicontext environment on top of additional power consumption due to reconfiguration.

2) Versus Baseline With SMC: Next, we compare the performance and cost of the NVMC-FPGA to the baseline that performs multicontext through SMC. The baseline maps all contexts into a single PCL and operates with SMC. These contexts are operated with a single global clock if they are accelerated simultaneously because VPR does not support the multiple clock regions and domains. Therefore, multiple contexts in the same PCL share a single global clock domain in our experiments. Experiments can be conducted through the area or the number of context constraints. In this article, we conducted SIMD experiments with the context constraint. All experimental results of the NVMC-FPGA are normalized to the baseline that performs multicontext through SMC.

Single Instruction Multiple Data Experiments: Fig. 17 represents experimental results of the critical path delay. The NVMC-FPGA shows a comparable critical path delay to the baseline, although the NVMC-LUT has a longer critical path delay than the baseline LUT. The reason is as follows. As the number of contexts increases, the baseline FPGA size increases because it has a single PCL. On the contrary, the NVMC-FPGA size is not increased because it maps multiple contexts on multiple PCLs. This size difference counteracts the latency gap between the SB-LUT and NVMC-LUT. The critical path delay of "bn4" in the NVMC-FPGA is much smaller due to the large FPGA size increase of the baseline. Consequently, the critical path delay of the NVMC-FPGA is 1.4% smaller on average when eight contexts are mapped.

However, it is not a fair performance comparison because the baseline can accelerate eight mapped contexts simultaneously at the cost of using the larger area. Therefore, we compare two FPGAs' performance using throughput per area (TPA), which can consider both performance and area difference.

Fig. 18 represents the normalized area of the NVMC-FPGA. The NVMC-FPGA area is much smaller than the baseline, although the baseline's LUT size is smaller. It is because the size of circuits mapped on the baseline increases as we use SMC. The baseline's area increase varies for each circuit because its size increase is bounded by different hardware elements. In the island-style routing architecture (Fig. 1), each hardware element increases differently when the FPGA size increases. LUTs, F/Fs, and DSPs increase proportionally to the square of the FPGA size. However, I/O blocks increase proportionally to the FPGA size. It means that when more circuits are mapped, the circuit whose size is bounded by the I/O blocks shows more area increase in the baseline. As a result, the NVMC-FPGA reduces the area usage by 89.2% compared to baseline on average when the eight contexts are mapped in an SIMD fashion.

Experimental results of the critical path delay and area for the NVMC-FPGA are used to compute TPA. As mentioned above, the baseline maps multiple contexts using SMC and can accelerate up to eight contexts simultaneously. In contrast, the NVMC-FPGA accelerates eight contexts with TMC. TPA is the fair metric to compare the performance of baseline (SMC) and the NVMC-FPGA (TMC).

The result is represented in Fig. 19. As shown in Fig. 17, the critical path delays of both FPGAs are very similar. Thus, TPA depends significantly on the area. For circuit bn4, TPA improved by 9$\times$ because both critical path delay and area are improved. For the circuit with no significant improvement in the area, the TPA is less than 1. On average, TPA has improved by about 15.3%.

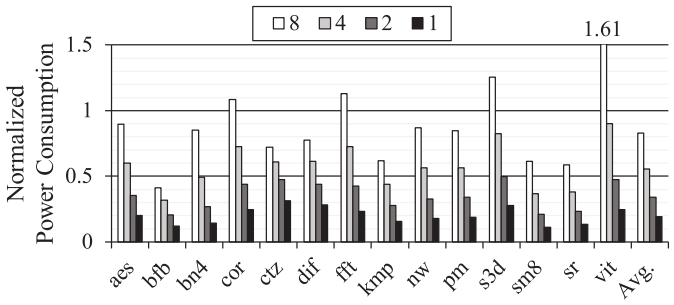

Fig. 20. Normalized power consumption with varying the number of contexts simultaneously activated (SIMD).

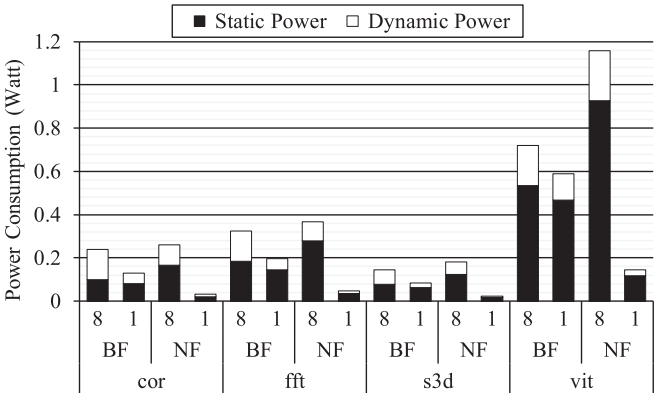

Fig. 21. Static and dynamic power consumption with eight contexts and one context simultaneously activated (SIMD).

In terms of power consumption, we measure the power consumption under the condition, where both the baseline and NVMC-FPGA produce the same throughput by duplicating the NVMC-FPGA. Both FPGAs map eight contexts, and we vary the number of activated contexts. That is, inactive contexts are in the idle state.

Fig. 20 shows the power consumption with respect to the number of simultaneously activated circuits. The case where 8 circuits are activated simultaneously is the worst-case scenario for the NVMC-FPGA because it cannot take advantage of power gating as all contexts are active. Nevertheless, the power consumption of most circuits decreases compared to the baseline. On average, the power consumption of the NVMC-FPGA is 11.2% smaller than the baseline when they accelerate 8 circuits simultaneously. Besides, cases with a smaller number of simultaneously activated circuits show larger improvements. If the number of simultaneously activated circuits is reduced, the NVMC-FPGA can save power consumption through power gating.

Fig. 21 represents the static and dynamic power consumption of selected workloads when 1 and 8 contexts are active for both FPGAs. The selected workloads are those with the larger power consumption of the NVMC-FPGA than the baseline when eight contexts are activated. "BF" and "NF" mean the baseline FPGA and the NVMC-FPGA, respectively. When the number of activated contexts is reduced from 8 to 1 in the baseline FPGA, the dynamic power consumption only decreases, while static power consumption does not. On the other hand,

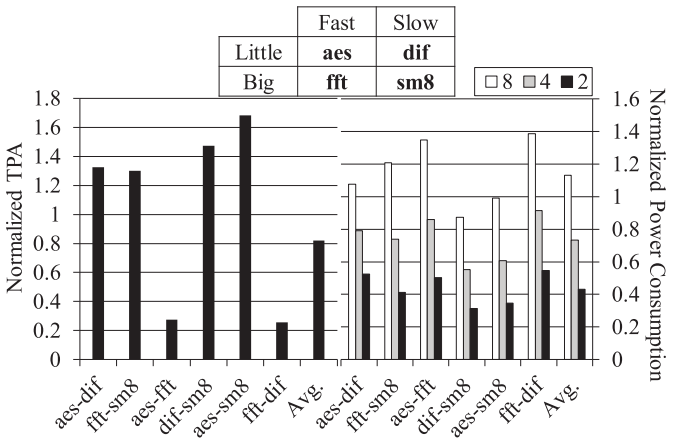

Fig. 22. Performance and power consumption when using TMC (MIMD).

both static and dynamic power consumption decrease in the NVMC-FPGA.

As a result, the power consumption of the NVMC-FPGA decreases more than the baseline by up to 80.7%, depending on the number of circuits activated simultaneously.

Multiple Instruction Multiple Data Experiments: MIMD experiments more closely reflect actual multithreading computing environments. To find the tendency of the performance and cost according to benchmarks' characteristics, the benchmarks are divided into two categories according to FPGA size and delay: 1) "Little" and "Big" in terms of FPGA size and 2) "Fast" and "Slow" in terms of delay. Four circuits are chosen to represent each category, and they are shown in Fig. 22.

First, we show experimental results only with TMC, represented in Fig. 22. In these experiments, both the NVMC-FPGA and baseline map four contexts for each circuit. When two circuits are in the same size category as "aes-dif" and "fft-sm8," the NVMC-FPGA improves TPA about 32% on average because there is no underutilizing for each PCLs in the NVMC-FPGA. When two circuits are in different size categories, the NVMC-FPGA may show less area efficiency because smaller circuits may not fully utilize the entire FPGA. That causes TPA degradation in cases of "aes-fft" and "fft-dif."

However, this degradation can be reduced by mapping several small circuits to a single PCL with SMC, which uses both TMC and SMC. Through SMC, the NVMC-FPGA maps 12 more contexts for each experiment aes-fft, fft-dif, "dif-sm8," and "aes-sm8," without any area overhead. Fig. 23 shows the performance and power consumption when the NVMC-FPGA performs with both TMC and SMC. When two circuits are in different size categories, TPA is improved as SMC increases space efficiency. In aes-fft, TPA degradation is significantly reduced from 72.4% to 32.4%. In fft-dif, TPA improvement has increased significantly from −74.2% to 10%. In dif-sm8 and aes-sm8, TPA is further enhanced. As a result, in the MIMD scenario, the NVMC-FPGA (with TMC and SMC) improves TPA by 58.5% on average.

In terms of the power consumption, experimental results show a similar tendency to results that produced in the SIMD

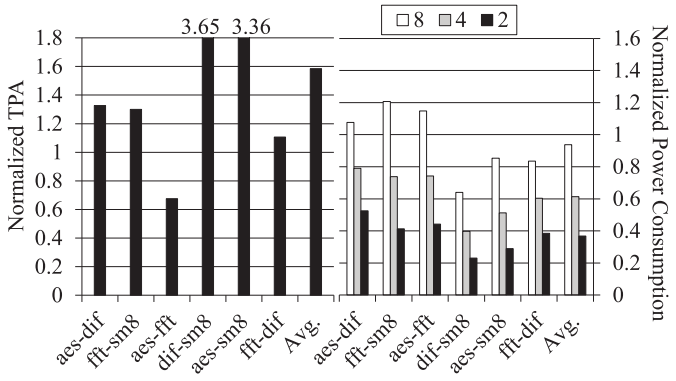

Fig. 23. Performance and power consumption when using both TMC and SMC (MIMD).

environment. As the number of simultaneously activated contexts decreases, the power consumption improves significantly in the case of the NVMC-FPGA due to partial power gating. SMC increases space efficiency and reduces the power consumption. In MIMD scenario, the NVMC-FPGA (with TMC and SMC) reduces the power consumption from 6.2% to 63.3%, depending on the number of circuits activated simultaneously.

V. Conclusion

We proposed NVMC-FPGA based on STT-MRAM. The NVMC-FPGA compensates the shortcomings of the previously proposed multicontext FPGAs by exploiting STT-MRAM-based LUT supporting multicontext. We proposed the NVMC-FPGA architecture and operation modes for the NVMC-FPGA that fully utilize the advantages of STT-MRAM. Moreover, we modified the FPGA CAD flow for the NVMC-FPGA.

In SIMD environment experiments, the NVMC-FPGA improves the performance by 15.3% than the conventional SRAM FPGA. The reduction of average power consumption is from 11.2% to 80.7%, depending on the number of simultaneously activated circuits through power gating. In MIMD environmental experiments that mimic the actual multithreading computing environment, the performance is improved by 58.5% on average, using both TMC and SMC. The reduction of average power consumption is from 6.2% to 63.3%, depending on the number of simultaneously activated circuits. Finally, we showed that our proposed NVMC-FPGA is adequate for the multithreading computing environment.

In the future, we will further improve the NVMC-FPGA to support the larger number of contexts and explore various energy-efficient computing methods. Furthermore, we will explore more diverse optimization criteria by modifying the design space exploration process of VPR and explore the optimal context mapping for the NVMC-FPGA under various real multithreaded workloads.

References

[1] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideovt, E. Bassous, and A. R. Leblanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Solid-State Circuits Soc. Newslett.*, vol. 12, no. 1, pp. 38–50, 2007.

[2] Y. S. Shao and D. Brooks, "Energy characterization and instruction-level energy model of Intel's Xeon Phi processor," in *Proc. IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, 2013, pp. 389–394.

[3] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, 2014, pp. 10–14.

[4] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Architect. (ISCA)*, 2014, pp. 13–24.

[5] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2016, pp. 1–13.

[6] P. K. Gupta, "Accelerating datacenter workloads," in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, 2017, p. 20.

[7] F.-L. Yuan, C. C. Wang, T.-H. Yu, and D. Marković, "A multi-granularity FPGA with hierarchical interconnects for efficient and flexible mobile computing," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 137–149, Jan. 2015.

[8] S. Jiang *et al.*, "Accelerating mobile applications at the network edge with software-programmable FPGAs," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 55–62.

[9] P. G. Mousouliotis and L. P. Petrou, "Software-defined FPGA accelerator design for mobile deep learning applications," in *Proc. Int. Symp. Appl. Reconfig. Comput.*, 2019, pp. 68–77.

[10] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, AND ASIC," in *Proc. IEEE Int. Conf. Field Program. Technol. (FPT)*, 2016, pp. 77–84.

[11] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. IEEE 26th Int. Conf. Field Program. Logic Appl. (FPL)*, 2016, pp. 1–4.

[12] P. K. Gupta, "Intel XEON+ FPGA platform for the data center," presented at the Proc. Workshop Reconfig. Comput. Masses Really, 2015.

[13] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and a cost model," *ACM Trans. Reconfig. Technol. Syst.*, vol. 4, no. 4, p. 36, 2011.

[14] S. Di Carlo, G. Gambardella, M. Indaco, P. Prinetto, D. Rolfo, and P. Trotta, "Dependable dynamic partial reconfiguration with minimal area & time overheads on Xilinx FPGAs," in *Proc. 23rd Int. Conf. Field Program. Logic Appl.*, 2013, pp. 1–4.

[15] A. DeHon, "Dynamically programmable gate arrays: A step toward increased computational density," in *Proc. 4th Can. Workshop Field Program. Devices*, vol. 8, 1996.

[16] W. Chong, S. Ogata, M. Hariyama, and M. Kameyama, "Architecture of a multi-context FPGA using reconfigurable context memory," in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp.*, 2005, p. 7.

[17] Y. Birk and E. Fiksman, "Dynamic reconfiguration architectures for multi-context FPGAs," *Comput. Elect. Eng.*, vol. 35, no. 6, pp. 878–903, 2009.

[18] K. Tatsumura, M. Oda, and S. Yasuda, "A pure-CMOS nonvolatile multi-context configuration memory for dynamically reconfigurable FPGAs," in *Proc. IEEE Int. Conf. Field Program. Technol. (FPT)*, 2014, pp. 215–222.

[19] T. R. Halfhill, "Tabula's time machine," *Microprocess. Rep.*, vol. 131, Mar. 2010.

[20] M. Gao *et al.*, "DRAF: A low-power DRAM-based reconfigurable acceleration fabric," *ACM SIGARCH Comput. Architect. News*, vol. 44, no. 3, pp. 506–518, 2016.

[21] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1524–1537, Jun. 2015.

[22] S. Senni, L. Torres, G. Sassatelli, A. Gamatie, and B. Mussard, "Exploring MRAM technologies for energy efficient systems-on-chip," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 6, no. 3, pp. 279–292, Sep. 2016.

[23] W. Zhao, E. Belhaire, C. Chappert, and P. Mazoyer, "Spin transfer torque (STT)-MRAM-based runtime reconfiguration FPGA circuit," *ACM Trans. Embedded Comput. Syst.*, vol. 9, no. 2, p. 14, 2009.

[24] Y. Guillemenet *et al.*, "MRAM based EFPGAs: Programming and silicon flows, exploration environments, MRAM current state in industry and its unique potentials for FPGAs," in *Proc. IEEE Int. Conf. Reconfig. Computing . FPGAs*, 2009, pp. 18–23.

[25] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*. New York, NY, USA: Now, 2008.

[26] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *Proc. ACM/SIGDA 8th Int. Symp. Field Program. Gate Arrays*, 2000, pp. 203–213.
[27] K. Jo, K. Cho, and H. Yoon, "Variation-tolerant and low power look-up table (LUT) using spin-torque transfer magnetic RAM for non-volatile field programmable gate array (FPGA)," in *Proc. IEEE Int. SoC Design Conf. (ISOCC)*, 2016, pp. 101–102.
[28] S. Paul, S. Mukhopadhyay, and S. Bhunia, "Hybrid CMOS-STTRAM non-volatile FPGA: Design challenges and optimization approaches," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2008, pp. 589–592.
[29] K. Cho, S. Lee, C. Lee, T. Yim, and H. Yoon, "Low power multi-context look-up table (LUT) using spin-torque transfer magnetic RAM for non-volatile FPGA," in *Proc. IEEE Int. SoC Design Conf. (ISOCC)*, 2017, pp. 107–108.
[30] W. Zhao, C. Chappert, V. Javerliac, and J.-P. Noziere, "High speed, high stability and low power sensing amplifier for MTJ/CMOS hybrid logic circuits," *IEEE Trans. Magn.*, vol. 45, no. 10, pp. 3784–3787, Oct. 2009.
[31] K. E. Murray *et al.*, "VTR 8: High performance CAD and customizable FPGA architecture modelling," *ACM Trans. Reconfig. Technol. Syst.*, vol. 31, no. 2, pp. 1–55, 2020.
[32] L. Zhang *et al.*, "Addressing the thermal issues of STT-MRAM from compact modeling to design techniques," *IEEE Trans. Nanotechnol.*, vol. 17, no. 2, pp. 345–352, Mar. 2018.
[33] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 3, pp. 288–298, Mar. 2004.
[34] Intel Corp. (2020). *Intel Stratix Series FPGAs and SoCs*. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/stratix-series.html
[35] J. Luu *et al.*, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," *ACM Trans. Reconfig. Technol. Syst.*, vol. 4, no. 4, p. 32, 2011.
[36] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2014, pp. 110–119.
[37] Xilinx. (2020). *Vivado High-Level Synthesis*. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html
[38] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research," *J. Pharmaceut. Biomed. Anal.*, vol. 22, no. 5, pp. 717–727, 2000.
[39] J. S. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 211–217.
[40] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "ODIN II—An open-source verilog HDL synthesis tool for CAD research," in *Proc. 18th IEEE Annu. Int. Symp. Field Program. Cust. Comput. Mach.*, 2010, pp. 149–156.
[41] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. Int. Conf. Comput.-Aided Verification*, 2010, pp. 24–40.
[42] W. Zhao and Y. Cao, "Predictive technology model for nano-CMOS design exploration," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 3, no. 1, p. 1, 2007.
[43] A. S. University. (2011). *Predictive Technology Model*. [Online]. Available: http://ptm.asu.edu
[44] C. Wolf. (2016). *Yosys Open Synthesis Suite*. [Online]. Available: http://www.clifford.at/yosys/s

Jeongbin Kim received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree.

His research interests include flash memory applications, nonvolatile memory, and high-performance system architectures.

Yongwoon Song (Student Member, IEEE) received the B.S. degree in computer science and engineering from Sogang University, Seoul, South Korea, in 2012, where he is currently pursuing the Ph.D. degree.

His research interests include application-specific memory architectures and embedded systems.

Kyungseon Cho received the M.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2018.

Her research interests include integrated solid-state devices and circuits.

Hyukjun Lee (Member, IEEE) received the B.S. degree in computer engineering from the University of Southern California at Los Angeles, Los Angeles, CA, USA, in 1993, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 1995 and 2001, respectively.

He is currently a Professor with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. His research interests include embedded systems, low-power design, and memory/storage architectures.

Hongil Yoon (Member, IEEE) received the B.S. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA, in 1991, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of Michigan, Ann Arbor, MI, USA, in 1993 and 1996, respectively.

He is currently a Professor with the School of Electrical and Electronic Engineering, Yonsei University, Seoul, South Korea. His research interests include low-voltage memory circuit and technology.

Eui-Young Chung (Member, IEEE) received the B.S. and M.S. degrees in electronics and computer engineering from Korea University, Seoul, South Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2002.

He is currently a Professor with the School of Electrical and Electronics Engineering, Yonsei University, Seoul. His research interests include system architecture and VLSI design, including all aspects of computer-aided design.