Towards Efficient Reconfiguration through Lightweight Input Inversion for MLC NVFPGAs

Huichuan Zheng, Mengying Zhao*, Hao Zhang, Yuqing Xiong, Xiaojun Cai, Zhiping Jia
School of Computer Science and Technology, Shandong University, China

***Abstract*—Nonvolatile field programmable gate arrays (NVFPGAs) have been proposed to address the challenges raised by artificial intelligence and big data related applications, since nonvolatile memories (NVMs) introduce advantages of high storage density, low leakage power, and high system robustness. In addition, multi-level cell (MLC), which can store multiple bits within one memory cell, further improves the logic density of NVFPGAs. However, the inefficient write operation of MLC NVM significantly increases the reconfiguration cost in aspects of energy, latency, and lifetime. In this paper, we focus on the reconfiguration cost of MLC LUTs in NVFPGA and propose a lightweight input inversion based scheme to reduce the reconfiguration cost. Inversion flexibility is defined and modeled for LUT inputs to guide the proposed scheme. We also discuss how the proposed scheme can be combined with other existing write reduction strategies. Evaluation shows the proposed scheme can reduce reconfiguration cost by 10.01% with negligible overhead.**

***Index Terms*—Field-programmable gate array (FPGA), nonvolatile memory (NVM), multi-level cell (MLC), reconfiguration cost.**

I. Introduction

Nowadays, field programmable gate arrays (FPGAs) have been widely applied in area of big data and artificial intelligence due to features of low power consumption and high reconfigurable flexibility. With continuously increasing scale of data, FPGAs are expected to provide higher computational and storage density [1]–[5]. Traditional FPGAs use static random-access memory (SRAM) to build up their programmable components, i.e., configurable logical blocks (CLBs), switch boxes (SBs), and connection boxes (CBs), which faces the challenges of limited density and high leakage power. As a promising solution, emerging nonvolatile memory (NVM), which has high density and near-zero leakage power, has been proposed to replace SRAM in FPGAs, leading to nonvolatile FPGAs (NVFPGAs), e.g., phase-change memory (PCM) based FPGA [6], spin-transfer torque magnetic memory (STT-RAM) based FPGA [7], and resistive random access memory (RRAM) based FPGA [8], [9]. Representative NVMs usually can work in either single-level cell (SLC) or multi-level cell (MLC) mode. Figure 1 shows an example of SLC STT-RAM and MLC STT-RAM. In MLC mode, two bits, called hard bit and soft bit are simultaneously stored in a single physical cell, leading to higher computational and storage density than SLC mode.

Though MLC NVFPGAs have higher storage density, they also have disadvantages. When reconfiguring, a new logic can be reconfigured into FPGA by rewriting contents of programmable components. Write operations in MLC NVM typically have higher latency and energy consumption than SLC, due to the complicated write process of hard bit [10]. Thus, the reconfiguration cost becomes more significant. Taking MLC STT-RAM for example, its reconfiguration latency and energy are 2.21 times and 2.98 times of SLC STT-RAM, respectively [11]. The situation will become even worse in scenarios with more frequent reconfigurations.

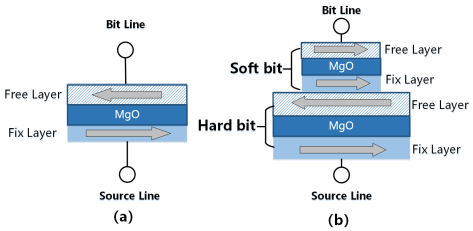

Fig. 1. Structures of (a) Single-Level Cell (SLC) and (b) Multi-Level Cell (MLC). In MLC STT-RAM, two logical bits can be stored in a single physical cell, leading to doubled storage density than SLC STT-RAM.

Since the cost of reconfiguration is directly determined by the amount of data to be written in order to configure NVFPGAs to the new logic, two categories of strategies are motivated to reduce reconfiguration cost of NVFPGAs. First, there are existing researches working on write elimination in NVM fields, such as differential write [12], Flip-N-Write (FNW) [13], which can be adapted to reconfiguration of NVFPGAs. Since there are significant differences in architecture of NVM and NVFPGA, these strategies need to be adjusted to fit into NVFPGAs. For example, FNW is proposed to flip "0" and "1" if the number of bits to write is higher than half of the total bits, so that the amount of bits to write is always limited to a half. In traditional NVM area, this can be conducted with granularity of a memory line. When applying FNW to NVFPGAs, we need to redefine the granularity, e.g., a look-up-table. More importantly, FNW needs extra bits to record whether it is flipped or not. In NVFPGA, the tag bits as well as logic to access these bits need to be integrated into the target configuration. The other category of strategy is to directly design mechanisms considering features of NVFPGAs. Considering typical FPGA architectures including, configurable resources of CLBs, SBs, and CBs as shown in Figure 2(a), we can compare data in each component between adjacent configurations, and try to reduce difference between them, e.g., maximize reused path [14], or eliminate bit flips in CLBs by input swapping [15].

This work was supported by the National Key Research and Development Program of China under Grant 2022YFB4502004. The corresponding author is Mengying Zhao. (e-mail: zhaomengying@sdu.edu.cn)

979-8-3503-4859-0/DATE24/© 2024 EDAA

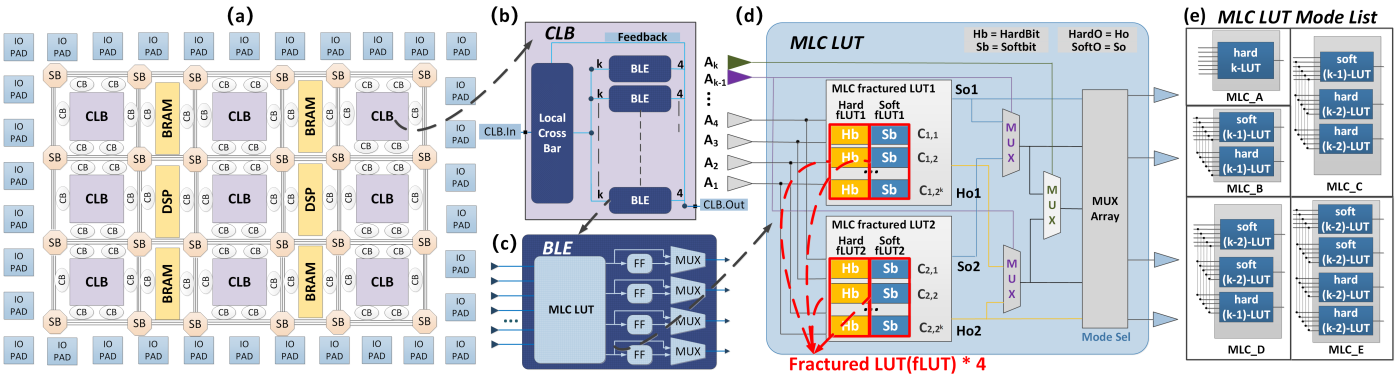

Fig. 2. The NVFPGA architecture proposed in [16], including: (a) FPGA model, (b) CLB model, (c) BLE model, (d) MLC LUT model, and (e) MLC LUT mode list.

When it comes to MLC NVFPGA, optimization of reconfiguration cost becomes more complicated, since MLC mode introduces hard bits into FPGAs, which have higher write energy and longer write latency than soft bits. So we need to differentiate soft and hard bits in reconfiguration cost models. More importantly, we should take the performance asymmetry into consideration since data redistribution with objective of minimizing writes may degrade system performance if hard bits dominate critical paths.

In this paper, we aim to design an input-inversion-based scheme to reduce the reconfiguration cost for MLC LUTs in NVFPGAs. The scheme exploits the inversion flexibility of LUT inputs to redistribute the content of LUTs. We also study how the proposed input-inversion strategy can be combined with existing write reduction techniques and present schemes to reduce area overhead of the combined design. The contributions of this paper are as follows.

- Define different types of inversion flexibility of LUT inputs and exploit the input inversion flexibility to reduce the reconfiguration cost.
- Give solutions of combining other write reduction technique into the proposed scheme for further cost reduction.
- Evaluate the proposed scheme and compare reconfiguration cost improvements with existing related works.

The remainder of this paper is organized as follows. Section II introduces the background and summarizes related works. Section III introduces the inversion flexibility of LUT inputs and proposes a lightweight input inversion based technique to optimize the reconfiguration cost. This section also discusses how to combine the proposed scheme with existing write reduction strategies. Section IV gives the evaluation results and discussions, followed by overhead analysis. Section V concludes this paper.

II. Preliminary and Related Work

A. Multi-Level Cell (MLC) Look-Up Table (LUT) in NVFPGA

Figure 2(a) shows a typical architecture of NVFPGA proposed in [16]. Components including configurable logical blocks (CLBs), switch boxes (SBs), connection boxes (CBs), and block random access memories (BRAMs) are built up with nonvolatile memories.

CLBs are main logical resources of FPGAs, which can implement combinational and sequential logics. As Figure 2(b) demonstrates, each CLB contains a cluster of basic logic elements (BLEs) along with a fully populated crossbar. As shown in Figure 2(c), BLE can be further decomposed into one MLC look-up table (LUT) with k inputs, some flip-flops (FFs), and several multiplexers (MUXs). Figure 2(d) shows the detailed structure of a k-input MLC LUT, which can be decomposed into two MLC fractured LUTs with k-1 inputs. Since fractured LUT works in MLC mode, it can be further divided into soft fractured LUT and hard fractured LUT. Thus, an MLC LUT contains four fractured (k-2)-input LUTs (fLUTs) built up with either 2^{k-2} hard bits or 2^{k-2} soft bits.

By configuring its mode selection signal, an LUT can implement either a big logical LUT or several smaller LUTs by combining fractured LUTs in different ways, which are listed in Figure 2(e). In mode MLC_A, all four fractured LUTs are combined together to form a k-input LUT. In mode MLC_B, hard fLUTs and soft fLUTs are combined in pairs to form a hard bit based (k-1)-input LUT and a soft bit based (k-1)-input LUT. In mode MLC_C, two soft fLUTs work together to form a soft bit based (k-1)-input LUT, while two hard fLUTs work independently. On the contrary, in mode MLC_D, two hard fLUTs work together while two soft fLUTs work independently. In mode MLC_E, each fLUT does its own job to carry a (k-2)-input logical LUT. Due to the performance difference, hard bit based LUTs have larger delay than soft bit based ones. As for an MLC_A LUT, its performance is dominated by those slow hard bits.

The reconfiguration cost of MLC LUT in NVFPGAs varies from each other due to inherent programming asymmetry as shown in Table I.

B. Write Reduction Techniques for NVMs

In traditional NVM area, there are plenty of strategies to achieve the goal of write reducing. Differential-write [12] performs comparisons between existed data and new data, and only re-writes those miss-matched cells. Flip-N-Write [13]

TABLE I
THE WRITE ENERGY AND LATENCY OF MLC STT-RAM

From (hard-soft) \ to (hard-soft)		00	01	10	11
00	Energy(nJ)	0	0.843	**2.502**	1.659
	Latency(Cycs)	0	25.31	**56.50**	31.19
01	Energy(nJ)	0.843	0	**2.502**	1.659
	Latency(Cycs)	25.31	0	**56.50**	31.19
10	Energy(nJ)	1.659	**2.502**	0	0.843
	Latency(Cycs)	31.19	**56.50**	0	25.31
11	Energy(nJ)	1.659	**2.502**	0.843	0
	Latency(Cycs)	31.19	**56.50**	25.31	0

encodes the data into whether regular form or inverted form to obtain the least bit-writes. CAFO [17] performs Flip-N-Write in both rows and columns to achieve more write reductions with less auxiliary bits required. Flip-Min [18] performs an one-to-many mapping for each dataword to a coset of vectors, so that the predefined vector with the least bit-writes can be selected when configuring. Since these techniques are not originally designed for NVFPGA, they need to be adjusted to fit to FPGA features.

There are also several techniques aim directly to NVFPGAs, which take existing logical implementations into considerations when generating target configuration. An input reordering technique is proposed to redistribute the content of LUTs [15]. It generates solutions with different input orders, in which LUT content is distributed differently. The solution that can minimize the number of writes will be adopted. A routing path reuse method is proposed to efficiently configure routing resources in NVFPGAs [14]. It proposes a reuse-aware routing algorithm to reduce the reconfiguration cost of switch boxes. However, these works are designed for SLC NVFPGA so that features introduced by MLC are not covered.

In this paper, we propose a lightweight input inversion based scheme to optimize the reconfiguration cost for MLC LUTs in NVFPGA.

III. LIGHTWEIGHT INPUT INVERSION FOR MLC NVFPGAS

In this section, we will introduce the proposed lightweight input inversion strategy for MLC FPGAs in detail.

A. Main Idea

In order to reduce reconfiguration cost, we follow the solution of keeping data in FPGA components unchanged as much as possible by encoding configuration data. However, complicated encoding tend to involve heavy overhead since all encoding and decoding logic need to be implemented using FPGA resources, too.

Thus, we study usage of input inversion, which is a lightweight encoding, in MLC NVFPGAs. Usually, LUTs dominate the reconfiguration, so we mainly focus on write reduction for LUTs. Specifically, we examine whether to invert inputs of LUTs, i.e., from 1/0 to 0/1, so that content in LUTs can be redistributed to minimize writes. We first analyze the inversion flexibility of inputs in MLC LUTs, and then propose inversion scheme to reduce reconfiguration cost. Finally, we discuss how the proposed scheme can be combined with existing write reduction strategies.

B. Inversion Flexibility (IF) of Inputs in MLC LUT

Before defining inversion strategy for NVFPGA, we can analyze what inversion means for MLC LUTs. Figure 3 shows an example of how input inversion redistributes the content of LUT. Each time an input is inverted, the content of LUT will be half-to-half shuffled. When input I_1 is inverted, cells are exchanged in the granularity of each two bits. While the subsequent inversion of the most significant input (MSI), i.e., I_3, swaps the content between hard bits and soft bits.

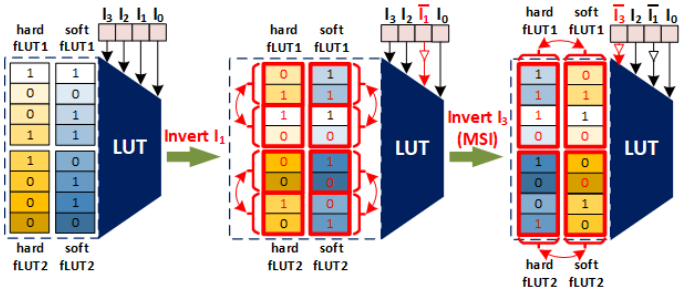

Fig. 3. An example of LUT input inversion.

However, for an MLC LUT, its content can not be redistributed freely because hard bits may be possibly redistributed to critical paths, leading to system performance degradation. Thus, we define inversion flexibility for LUT inputs to guide the inversion process. Inversion flexibilities of an LUT's inputs are correlated to the mode where the LUT is working in.

Assuming the 4-input LUT in Figure 3 is working in mode MLC_A of Figure 2(e), all bits belong to a whole 4-input logical LUT. Since the performance of this LUT is already dominated by hard bits, all inputs can be inverted without prolonging the latency of the LUT. Thus, we define inputs in this case have high inversion flexibility.

For MLC_C, it can be regarded to contain one 3-input LUT with soft bits, and two 2-input LUTs with hard bits. If the MSI, i.e., I_3, is inverted, contents of soft bits and hard bits will be exchanged. Assuming the 3-input logical LUT with soft bits is in the critical path, such exchange will degrade the system performance. Thus, MSI should be considered as a low flexibility input. As for the other inputs, since their inversion lead to content swapping within hard bits or soft bits, inverting them will not causing delay changes for the implemented logical LUTs. So these inputs can be considered as inputs with high inversion flexibility.

Similarly, for LUTs working in mode MLC_B, MLC_D, and MLC_E, the inversion flexibility of their MSIs should also be defined as low, since their inversion will exchange the contents of hard bits and soft bits, leading to potential performance degradation.

To sum up, we propose to categorize inputs into two types with high or low inversion flexibility, which will be used for input inversion decision. Those inputs that potentially affect the delay of their LUTs are marked as low inversion flexibility. While the other inputs are considered as high inversion flexibility inputs since inverting them will not degrade the system performance.

C. IF-aware Input Inversion

Based on the input inversion flexibility, we propose an input inversion scheme to reduce reconfiguration cost. Figure 4 shows the overall design of our proposed inversion scheme along with the widely used VTR synthesis flow [19]. The proposed scheme will be performed after physical synthesis has been finished. It takes the original output file from VTR synthesis flow and gives a cost-optimized output file.

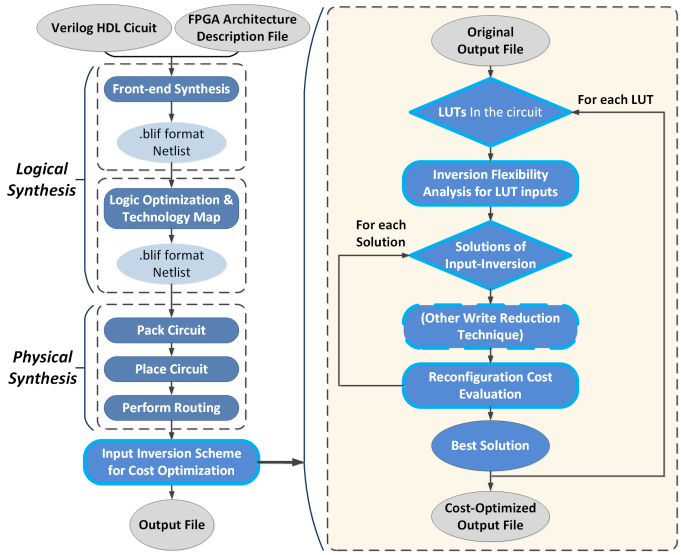

Fig. 4. The proposed input inversion scheme and the VTR CAD flow [19].

As the outer loop of Figure 4 shows, for each LUT, the scheme will firstly analyze the flexibility of its inputs. Then we need to decide whether to invert them or not. As an observation, according to statistics, except from MLC_A LUTs, there are 98% of LUTs in the critical path are soft LUTs, which means redistribution between soft and hard bits have quite large possibility to prolong the critical path, i.e., degrade the system performance. So we tend to avoid inverting inputs with low IF and decide which inputs to invert for those with high IF. Various solutions, in which inputs are inverted differently, are generated through the analysis process, which will be checked in the inner loop as shown in Figure 4.

Each input inversion solution is represented by a vector of $Input\_Tags$, in which each tag identifies whether the corresponding input should be inverted or not. The LUT's content will be redistributed based on the $Input\_Tag$ vector of the corresponding solution. We can evaluate the reconfiguration cost of each inversion solution. Equation 1 shows reconfiguration energy cost as an instance.

$$Cost_{LUT} = e_H * (d_{H1} + d_{H2}) + e_S * (d_{S1} + d_{S2}) \quad (1)$$

Here d_{H1}, d_{H2}, d_{S1}, and d_{S2} are the numbers of cells to be reprogrammed in each fractured LUTs, i.e., hard fLUT1, hard fLUT2, soft fLUT1, and soft fLUT2, respectively. While e_H and e_S are the energy consumption of programming a hard bit cell and a soft bit cell, respectively, which can be checked in Table I. After checking all inversion solutions on inputs with high IF, the solution whose $Input\_Tag$ vector can minimize the LUT cost will be adopted.

When configuring the FPGA, the redistributed LUT contents will be loaded along with the adopted $Input_Tag$s. At runtime, actual input can be generated by performing excursive-OR between original input and the corresponding $Input\_Tag$, as shown in Figure 5.

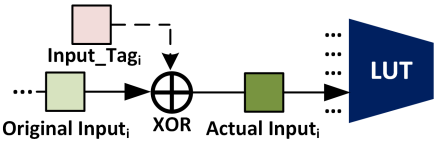

Fig. 5. The actual input generation process.

D. Case Study: Combining with FNW

Since the proposed input inversion scheme only distributes the LUT content through input manipulating, it has the potential to combine with other write reduction techniques on LUT content to further optimize reconfiguration cost.

In this section, we integrate the widely-used data encoding technique, i.e., FNW, into the proposed scheme. As mentioned before, FNW should be adjusted to adapt to MLC LUTs.

Since the MLC LUT structure can be considered as four fractured LUTs with their own outputs, the granularity of FNW is set as fractured LUT. We decide whether an LUT will be flipped or not based on which form has less reconfiguration cost, which is determined by the number bits to be rewritten. Different from SLC NVMs, flipping a hard bit in STT-RAM based MLC LUT will overwrite its corresponding soft bit [11]. Thus, the number of bits to be rewritten in a soft fractured LUT is related to the corresponding hard fractured LUT. Table II summarizes how to update the counters of these bits, i.e., d_H and d_S, during bit counting for STT-RAM based MLC LUT.

TABLE II
BIT COUNTING IN MLC LUT

Hard bit	Soft bit	Action
Match	Match	/
Match	Miss-match	$d_S + 1$
Miss-match	Match	$d_H + 1$ $d_S + 1$
Miss-match	Miss-match	$d_H + 1$

For each LUT, we use a vector of four $fLUT\_Tags$ to identify whether each fractured LUT is flipped or not. The $fLUT\_Tag$ will be set to "1" if its corresponding fractured LUT is flipped and vice versa. In an MLC LUT, several fractured LUTs may be packed up to implement a larger logical LUT. For example, in an MLC_A LUT, where four fractured LUTs are implementing a k-input logical LUT, they should keep the same $fLUT\_Tags$.

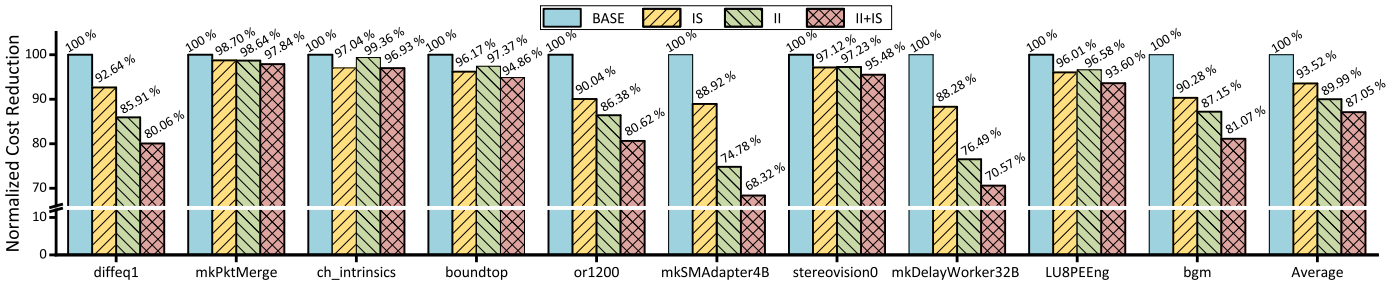

Fig. 7. Normalized reconfiguration costs of different schemes on various benchmarks.

After the FNW process, the output of this LUT is encoded, i.e., flipped or not, as shown in Figure 6(a). To derive the actual output, the encoded output of LUT1 is immediately decoded by performing exclusive-OR with the corresponding $fLUT\_Tag$. Then the output will be took by LUT2 as an input after another time of exclusive-OR for input inversion. Noticed that both the output-decoding process and the input-inverting process own their independent auxiliary logics, though they are serving exactly the same signal.

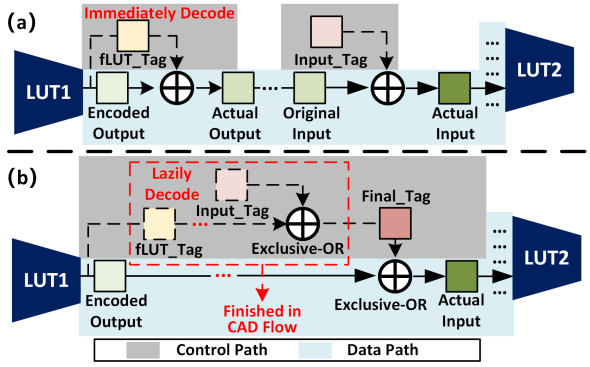

Fig. 6. The decoding process of the proposed scheme. (a) Decoding immediately using independent decode logics; (b) Decoding lazily using shared logics.

To further reduce the overhead, we decode the output signal lazily till it is took as an input of another LUT, i.e., LUT2 in Figure 6(b). In this manner, the decoding logic can be covered with the auxiliary logic of the aforementioned input-inversion technique. As Figure 6(b) shows, the $fLUT\_Tag$ will be exclusive-OR-ed with the $Input\_Tag$ of the assigned input pin to generate the $Final\_Tag$. The generation process of $Final\_Tag$ can also be accomplished during the offline synthesis stage, result in less runtime overheads.

After the $Final\_Tag$ is obtained, the encoded output will be transformed into the actual form. In this way, FNW shares the auxiliary logic of the proposed scheme, which introduces less overhead compared with using their own logics.

IV. Experimental Evaluation

In this section, we will evaluate the proposed reconfiguration scheme and report the cost reduction. Besides, we analyze the overhead of the proposed scheme and give discussions.

A. Experiment Setup

In evaluation, the proposed reconfiguration scheme is tested on NVFPGA architecture that proposed in [16], which uses STT-RAM based MLC LUTs to build up its CLBs. This FPGA architecture[1] is based on Altera Stratix IV device.

Our proposed scheme is integrated into the widely-used VTR CAD flow [19], which will be activated after the routing stage.

In order to show the efficacy in reconfiguration cost reduction, we implement several reconfiguration schemes for comparison.

- **BASE**: Reconfigure the FPGA with no optimization.
- **IS (Input-Swapping)**: Reconfigure the FPGA with the LUT input swapping technique proposed in [15].
- **II (Input-Inversion)**: Reconfigure the FPGA with the proposed input inversion technique.
- **II+IS**: Reconfigure the FPGA with the proposed input inversion technique and the input swapping technique.
- **FNW**: Reconfigure the FPGA using FNW [13].
- **II+FNW**: Reconfigure the FPGA with both the proposed input inversion technique and FNW.

A series of VTR benchmarks are adopted for evaluation, as listed in Table III, which have different complexities in terms of scale of circuits. We reconfigure the NVFPGA from one benchmark to another according to the sequence listed in Table III, and report the reconfiguration energy as cost.

TABLE III
VTR BENCHMARKS [19]

Benchmark	diffeq1	mkPkt Merge	ch_intrinsics	boundtop	or1200
CLBs	28	16	32	195	242
LUTs	379	217	399	2670	2685
Benchmark	**mkSM Adapter4B**	**stereo vision0**	**mkDelay Worker32B**	**LU8PEEng**	**bgm**
CLBs	145	706	398	2026	2950
LUTs	1744	11113	5079	20178	29177

B. Reconfiguration Cost Reduction

Figure 7 shows the normalized reconfiguration cost of BASE, IS, II, and II+IS. In average, the proposed input-inversion based technique achieves 10.01% cost reduction compared with BASE, and 3.53% cost reduction than IS. Since input-inversion and input-swapping explore totally different design spaces, they are orthogonal and can work cooperatively. As a result, II+IS achieves 12.95% of cost reduction, which is higher than either II or IS.

In the reported experimental results, noticed that both II and IS achieve low cost reductions on benchmark "mkPktMerge"

[1]The architecture file is *k6_frac_N10_mem32k_40nm.xml* (height: 100 width: 100), which has 7500 CLBs and 208 BRAMs in total.

and “ch_intrinsics”. That is because benchmarks are configured in sequence in our experiment. The NVFPGA is reconfigured to “mkPktMerge” from “diffeq1”. Here “diffeq1” has quite simple logic and uses comparatively a small number of LUTs. While reconfiguring the platform to “mkPktMerge”, a large amount of LUTs are empty, i.e., all bits are initialized to “0”. Thus, performing input inversion to redistribute content has little impact on reconfiguration cost in these LUTs. It is a similar case for “ch_intrinsics”. To validate the above analysis, we conduct another group of evaluations by changing the contents of LUTs before reconfiguration to random data. As shown in Figure 8, the proposed scheme delivers 20.44% and 15.30% cost reductions for “mkPktMerge” and “ch_intrinsics”, respectively. In average, the proposed scheme achieves 16.94% reduction of reconfiguration cost, when the existing contents of LUTs are initialized randomly.

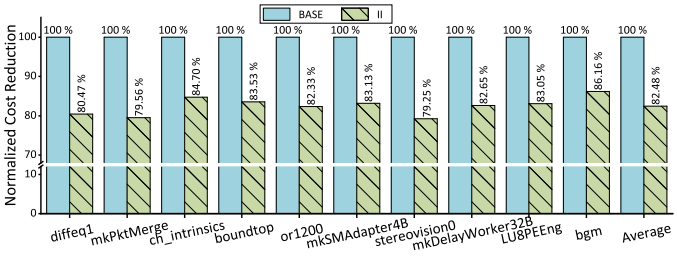

Fig. 8. Normalized reconfiguration costs of randomly initialized LUTs.

We also test the results by combining the proposed II and FNW. As shown in Figure 9, II+FNW achieves 8.89% more cost reduction than FNW itself.

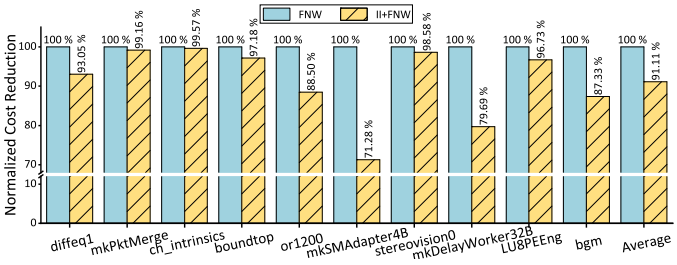

Fig. 9. Normalized reconfiguration costs of combining the proposed scheme with FNW.

C. Overhead Analysis

Since our scheme will not change the solution of physical synthesis, the overhead of our reconfiguration scheme is mainly introduced by the auxiliary logics.

As shown in Figure 6(b), an exclusive-OR gate is assigned to each input of an LUT. We evaluate the area and performance overhead using COFFE [20]. Compare with the architecture proposed in [16], the average area overhead for an LUT is 5.67%. While the delay of auxiliary logic increases the length of critical path by 1.80%, which is the performance overhead of the proposed scheme. The storage overhead, which is introduced to store tag bits, is 5.45% in evaluation.

V. Conclusion

In this paper, we propose a reconfiguration cost optimization scheme for MLC LUT in nonvolatile FPGAs, which is based on lightweight input inversion. Considering the features of MLC LUTs, we first define and model the inversion flexibility for LUT inputs. Then, an input inversion based scheme is proposed to minimize the reconfiguration cost following the guidance of inversion flexibility. We also study how to combine existing write reduction techniques and give an example based on the widely-used Flip-N-Write technique. At last, we conduct experiments to compare the cost reduction with different schemes and give evaluations about overhead. The experimental result shows a reconfiguration cost reduction of 10.01% on average compared with traditional reconfiguration.

References

[1] H. Sharma, J. Park, D. Mahajan *et al.*, “From high-level deep neural models to FPGAs,” in *MICRO*, 2016, pp. 17:1–17:12.

[2] S. Liu, C. Zeng, H. Fan *et al.*, “Memory-efficient architecture for accelerating generative networks on FPGA,” in *FPT*, 2018, pp. 30–37.

[3] C. Wang, L. Gong, Q. Yu *et al.*, “DLAU: A scalable deep learning accelerator unit on FPGA,” *IEEE TCAD*, vol. 36, no. 3, pp. 513–517, 2017.

[4] A. Putnam, A. M. Caulfield, E. S. Chung *et al.*, “A reconfigurable fabric for accelerating large-scale datacenter services,” *Commun. ACM*, vol. 59, no. 11, pp. 114–122, 2016.

[5] A. Tauhid, L. Xu, M. Rahman, and E. Tomai, “A survey on security analysis of machine learning-oriented hardware and software intellectual property,” *Elsevier High-Confidence Computing*, pp. 100–114, 2023.

[6] Y. Chen, J. Zhao, and Y. Xie, “3D-nonFAR: three-dimensional non-volatile FPGA architecture using phase change memory,” in *ISLPED*, 2010, pp. 55–60.

[7] S. Paul, S. Mukhopadhyay, and S. Bhunia, “Hybrid CMOS-STTRAM non-volatile FPGA: design challenges and optimization approaches,” in *ICCAD*, 2008, pp. 589–592.

[8] S. Tanachutiwat, M. Liu, and W. Wang, “FPGA based on integration of CMOS and RRAM,” *IEEE TVLSI*, vol. 19, no. 11, pp. 2023–2032, 2011.

[9] H. P. Wong, H. Lee, S. Yu *et al.*, “Metal-oxide RRAM,” *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[10] Y. Chen, X. Wang, W. Zhu *et al.*, “Access scheme of multi-level cell spin-transfer torque random access memory and its optimization,” in *MWSCAS*. IEEE, 2010, pp. 1109–1112.

[11] X. Chen, N. Khoshavi, J. Zhou *et al.*, “AOS: adaptive overwrite scheme for energy-efficient MLC STT-RAM cache,” in *DAC*, 2016, pp. 170:1–170:6.

[12] B. C. Lee, P. Zhou, J. Yang *et al.*, “Phase-change technology and the future of main memory,” *IEEE Micro*, vol. 30, no. 1, p. 143, 2010.

[13] S. Cho and H. Lee, “Flip-n-write: a simple deterministic technique to improve PRAM write performance, energy and endurance,” in *MICRO*, 2009, pp. 347–357.

[14] Y. Xue, P. Cronin, C. Yang, and J. Hu, “Routing path reuse maximization for efficient NV-FPGA reconfiguration,” in *ASP-DAC*, 2016, pp. 360–365.

[15] Y. Xue, P. Cronin, C. Yang, and J. Hu, “Fine-tuning CLB placement to speed up reconfigurations in NVM-based FPGAs,” in *FPL*, 2015, pp. 1–8.

[16] K. Liu, M. Zhao, L. Ju *et al.*, “Applying multiple level cell to non-volatile FPGAs,” *ACM TECS*, vol. 19, no. 4, pp. 27:1–27:22, 2020.

[17] R. Maddah, S. M. Seyedzadeh, and R. G. Melhem, “CAFO: cost aware flip optimization for asymmetric memories,” in *HPCA*, 2015, pp. 320–330.

[18] A. N. Jacobvitz, A. R. Calderbank, and D. J. Sorin, “Coset coding to extend the lifetime of memory,” in *HPCA*, 2013, pp. 222–233.

[19] K. E. Murray, O. Petelin, S. Zhong *et al.*, “VTR 8: High-performance CAD and customizable FPGA architecture modelling,” *ACM TRETS*, vol. 13, no. 2, pp. 9:1–9:55, 2020.

[20] C. Chiasson and V. Betz, “COFFE: fully-automated transistor sizing for FPGAs,” in *FPT*, 2013, pp. 34–41.