Soft Error Mitigation for Deep Convolution Neural Network on FPGA Accelerators

Wenshuo Li[1], Guangjun Ge[1], Kaiyuan Guo[1], Xiaoming Chen[2], Qi Wei[1], Zhen Gao[3], Yu Wang[1], Huazhong Yang[1]
[1]Department of Electronic Engineering, BNRist, Tsinghua University, Beijing, China
[2]State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS, Beijing, China
[3]Tianjin International Engineering Institute, Tianjin University, Tianjin, China
[1]e-mail: yu-wang@tsinghua.edu.cn

Abstract—**Convolution neural networks (CNNs) have been widely used in many applications. Field-Programmable Gate Array (FPGA) based accelerator is an ideal solution for CNNs in embedded systems. However, the single event upset (SEU) effect in FPGA device may have a significant influence on the performance of CNNs. In this paper, we analyze the sensibility of CNNs to SEU and present a fault-tolerant design for CNN accelerators. First, we find that SEU in processing elements (PEs) has the worst effects on CNNs since it produces proportional errors and will not get refreshed. Furthermore, it is indicated that the large positive perturbation contributes almost all of the performance loss. Based on such observations, we propose an error detecting scheme to locate incorrect PEs and give an error masking method to achieve fault-tolerance. Experiments demonstrate that the proposed method achieves similar fault-tolerant performance with the triple modular redundancy (TMR) scheme while the overhead is much lower than it.**

I. Introduction

Convolution neural networks (CNNs) have been approved to be an excellent solution for various applications. CNNs usually involve massive computations and memory accesses, and thus have high demands on the energy efficiency of the computing platform. FPGA-based CNN accelerators (e.g., [1], [2]) have been widely studied to offer high performance with low power consumption. Moreover, they can be reconfigured to realize a dedicated hardware architecture for a particular CNN.

In recent years, commercial off-the-shelf (COTS) devices are increasingly utilized in spacecraft due to their high performance [3]. However, it is well-known that particles in the radiation environment can easily cause soft errors like Single Event Upset (SEU), which mainly occurs in the memory and flip-flops and leads to logic flip [4]. These faults may result in severe performance degradation for CNN accelerators. Hence, it is important to study the soft error issue in FPGA-based CNN accelerators in the space environment.

In the literature, there have been many works concerning the reliability of FPGAs. Refs. [5], [6], [7] discuss the optimal Triple Modular Redundancy (TMR) of FPGA critical logic. Besides, several logic resynthesis techniques [8], [9], [10] are proposed to mitigate SEU faults from the view of synthesis tools. Recently, some researches [11], [12], [13] on FPGA-based CNN accelerators are conducted to evaluate the influence of SEUs. However, these works mainly focus on naive neural networks with simple fully-connected layers and concern more about streaming architecture. All of these works have not concentrated on the modern FPGA-based CNN accelerators with instruction-driven architecture (ISA), which has massive parameters and complicated computing units.

In this paper, we analyze the SEU faults and propose a novel scheme to detect and eliminate the SEU faults. Our contributions are as follows.

- We give a detailed analysis of the possible faults of processing elements (PEs) and RAMs caused by SEUs. It shows the effects of faults in PEs are more severe than that in data buffers.
- We conduct a software simulation of the influence of bit-flip errors, and find that the errors leading to a large positive perturbation are more critical.
- We propose an error detecting design for PE array by making use of the free cycles of CNN accelerators. We also propose an error masking method to handle faults.
- Our proposed method achieves similar fault-tolerant performance with TMR while the overhead is much lower.

II. Preliminary

A. General Fault-tolerant Methods

Triple modular redundancy is a commonly used fault-tolerant approach. It triples the protected module and uses a voter to get the majority of votes as the final output. TMR can only mask the faults instead of eliminating them. So scrubbing is generally used to reload parts of the configuration of an FPGA to correct the fault bits. Recently, partial dynamic reconfiguration [6], [14], [7] is widely studied, which could correct fault bits without interrupting the process.

B. Neural Network Accelerator

FPGA-based CNN accelerators can be divided into two types [15]: Streaming Architectures (SAs) and Single Computation Engines (SCEs). SAs directly map layers to different blocks of an FPGA to perform computations easily and efficiently. Existing works [11], [12], [13] concentrate more on the SEUs of SAs. However, since CNN architectures are developing rapidly, SCEs [16], [2], [17] are more and more popular now. With ISAs, they can handle different neural network architectures and parameters without reconfiguring the FPGA. For SCEs, the core module is the PE, which deals with almost all the computations of the accelerator.

The accelerator usually has $(N \times H \times W)$ PEs, where N, H, W are parallelism parameters and refer to the number of output channels, the height and width of feature map in the convolution layer, respectively. That is to say, $N \times H \times W$ calculations are taking place simultaneously.

C. Fault Model

The calculations of CNNs mainly contain addition and multiplication operations. Generally, adders are implemented with look-up tables (LUTs) and multipliers are implemented with DSP slices in FPGA. There are some configuration bits in LUTs and DSPs to determine their functions. Besides, there are some flip-flops (FFs) to store the immediate results. If SEUs occur in the configuration bits, the wrong bits would not be refreshed like that in the FFs. A PE with wrong bits would handle $\frac{1}{\#PE}$ of the total computations, and spread the errors into these results.

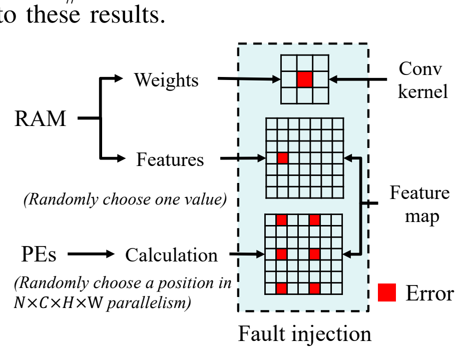
RAM
Weights
Features
(Randomly choose one value)
PEs
Calculation
(Randomly choose a position in N×C×H×W parallelism)
Conv kernel
Feature map
Error
Fault injection

Fig. 1. Fault injection of PEs and RAMs.

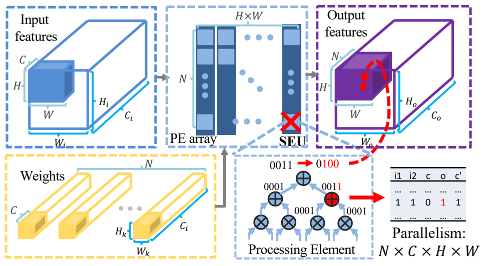

Fig. 2. How faults in PEs influence results.

Like Figure 2 shows, in every cycle, the accelerator performs convolution operations on $C \times H \times W$ input features with $N \times C \times H_k \times W_k$ weights. These calculations are mapped onto $N \times H \times W$ PEs. For example, one fault occurs in LUT and the output of the adder flips from 0 to 1. So there is one incorrect value in each $N \times H \times W$ output features. Actually, this fault can be any output bit of the adders flipped or the configuration of DSPs invalid. The former equals to the original results plus/minus 2^n, where n represents the position of the bit in the fixed-point data. The latter refers to a fault of the calculation mode. In our experiments, we directly inject faults to one of the $N \times H \times W$ outputs for simplicity.

It should be noted that our work concentrates on the user level. So we can only realize our design at the register transfer level (RTL), not at the gate level or inside the IP cores.

III. Simulation Experiments

A. Experiments Setup

We train MobileNetV2[18], ResNet-20[19], and GoogLeNet[20] on the CIFAR-10[21] dataset to conduct the experiments. To fit the implementation of FPGA-based CNN accelerator, we use 8-bit signed fixed-point integers to represent the weights and feature maps. The accuracies of various neural network models are shown in Table I.

TABLE I
Accuracy of 8-bit models on CIFAR-10

Model	MobileNetV2	ResNet-20	GoogLeNet
Baseline	92.12%[1]	91.25%[19]	93.64%[2]
8-bit fixed-point	92.02%	93.82%	92.53%

B. Comparison of Different Bits

To compare the influences of different injection positions, we perform experiments on PE faults and take an 8-bit adder as an example. We carry out the simulation with $16 \times 8 \times 1$ PEs. Since there are four types of input patterns in LUTs, we simply assume that the affected value has a 25% probability of being modified. The results are shown in Figure 3(a). Numbers on the x-axis represent the values we add to the original values.

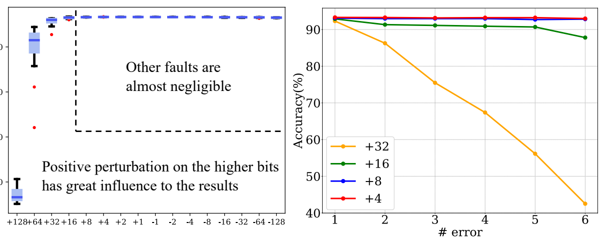

Fig. 3. (a) Influence of different injected bits. (b) Multiple errors injection into the lower bits.

From the results, we can see that the influence of positive perturbation is more significant, and the larger the perturbation, the higher the impact. We further inject multiple errors into the lower bits, and find that the lower four bits still cannot influence the results as Figure 3(b) shows. So the most important thing to design fault-tolerant architectures is to solve the large positive perturbation.

C. Comparison of PEs and RAMs

We inject the maximum disturbance to the original value to compare faults of PEs and RAMs in the worst cases. That is to say, we add 2^{n-1} to the chosen value, in which n represents the data width.

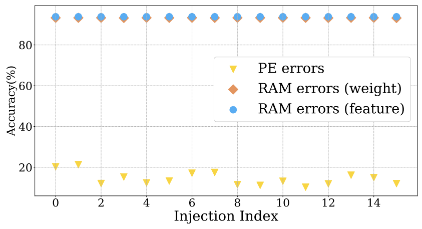

Fig. 4. Comparison of different types of fault injection

The results are shown in Figure 4, where injection index refers to the index of experiments since we randomly choose

[1]Our implementation.
[2]From https://github.com/conan7882/GoogLeNet-Inception-tf

positions to inject faults. And we can see that there is a limited influence of one single weight fault or feature fault on classification accuracy for ResNet-20 on CIFAR-10. While once the CNN inference encounters fault propagation caused by PEs' faults, the classification results will be affected dramatically.

IV. Fault Tolerant Design on PE Arrays

Based on the above fault injection analysis, conclusions can be drawn that FPGA-based CNN accelerators are sensitive to the persistent faults in PE. Thus in this section, we will introduce our fault-tolerant design on PE arrays.

A. Free Cycles of CNN FPGA Accelerators

To the best of our knowledge, there is no FPGA-based CNN accelerator that can reach its peak computing capacity. Actually, the average utilization ratio is usually below 85% [22]. It means the hardware resources have scheduling redundancy in CNN accelerators, i.e., the computing units have free cycles during inference. Some specific examples describing free cycles of PEs are as follows:

- Padding processing: Padding processing is an essential operation for most convolution layers. During padding processing, not all of the $H \times W$ PE groups are occupied.
- Residual edge operations: Since the dimension of a CNN layer cannot always be an integer multiple to the hardware parallelism, there will be free cycles for the residual edge operations. But such a situation does not always exist.
- Memory bandwidth mismatch: Though well designed CNN accelerator has considered the matching between memory bandwidth and computing throughput, it is still impossible to match them all through. For data-intensive layers, such as fully-connected (FC) layers, computing units will hang up until all the required data are loaded.

With different parallelism parameters, we have tried to evaluate the PEs' average free cycles during a single inference for various CNN models on CIFAR-10, as shown in Table II. In Table II, only free cycles introduced by padding processing are considered, since they only depend on the model size and thus are more stable. Based on the observation that PEs have plenty of free cycles, we propose an error detecting scheme, which can locate the PE with persistence errors.

TABLE II
Free cycles from Padding of PEs

Model(N,C,H,W)	MobileNetV2	ResNet-20	GoogLeNet
(8,8,1,4)	27584	94464	198400
(16,16,1,8)	6816	23680	49600

B. Error Detecting Schemes for PE Arrays

In this part, we first analyze the problems of PE error detecting. Then we give a traversal algorithm and corresponding hardware structure for error detecting.

1) Problems of PE Error Detecting: The adders mainly consist of LUTs and CARRYs [23], as shown in Figure 5. But for multipliers, it is more complicated since multipliers can be implemented by DSP slices, LUTs or DSP-LUT mixed in Xilinx FPGAs. For adders and multipliers, logic flipping in LUTs and DSP configuration ROMs will introduce persistent errors as they are configured at the moment of loading program file and will not get refreshed normally. For the DSP configuration ROMs, error detecting is easier, since the error will change DSP cores to different functions. But for LUTs, the problem is more difficult. It is known that a LUT realizes its function by storing all possible logical results of the inputs in advance. Soft errors occur in a LUT will only tamper one of these results. Therefore, errors can be detected only if we trigger the exact input combination, otherwise the error will be masked.

2) Traversal Error Detecting Algorithm: We design a traversal detecting algorithm for PE arrays. For simplicity, we divide the traversal into two parts, to detect multipliers and adders errors separately. To check the multipliers, we could simply set one of the inputs as 1, and traverse the other input. Full traversal of multipliers requires 2^n cycles, n represents the data width of fixed-points. To check the LUTs in adders, since the LUTs are only used as XOR function, there are only four possible error patterns. That is to say, we can finish a traversal of an adder in four cycles. For an m inputs adder-tree, there are $m-1$ adders in $log_2(m)$ layers. We can traverse the adder-tree layer-by-layer: After the adders in the lower layer are checked, we can traverse the adders in the higher layer by setting the output of lower adders as zeros, like Figure 6 shows.

The total overhead of traversal is $2^n + 4 \times log_2(m)$ cycles. As we mentioned in Section IV-A, this overhead can be confined within the free cycle quantity easily.

3) Structure of Error Detecting on PEs: In this part, we will introduce the hardware structure of our proposed error detecting scheme.

As shown in Figure 7, we add a PE Group Enable signal to control the input data of PE Array Group. If the PE Array Group is not in its free cycles, it would get data from CNN Data Buffer. While the PE Array Group is free, we load predefined check data to detect errors. It should be noted that we would not store the correct results of these check data. Instead, we just use a comparator to compare the results of different PEs and regard the minority as errors. In case that the free cycles of PEs are not balanced, there are extra data paths and multiplexers to rearrange PEs.

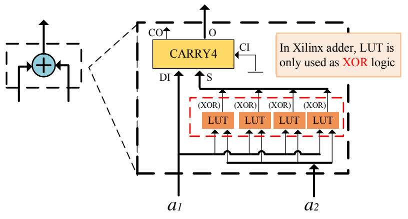

Fig. 5. Structure of Adder in Xilinx FPGA [23].

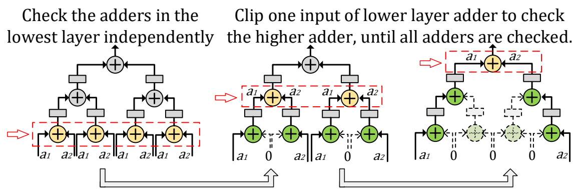
Check the adders in the lowest layer independently
Clip one input of lower layer adder to check the higher adder, until all adders are checked.
a1
a2
0

Fig. 6. Layered detecting for the adder-tree

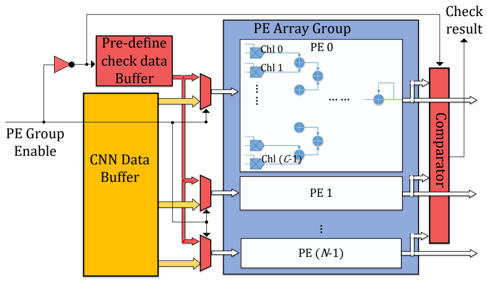

Fig. 7. Error Detecting for PE Group

V. Results

A. Simulation of Error Masking Capacity

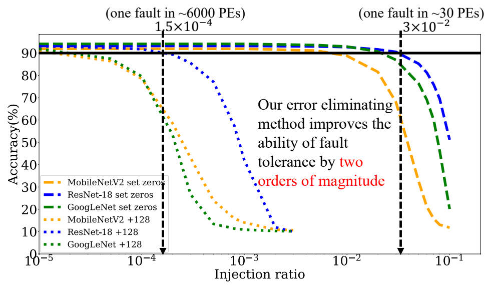

Fig. 8. Accuracies of random 0-injection and +128-injection

When it is not convenient to reconfigure LUTs immediately, we recommend to set the outputs of LUTs with SEU as zeros to mitigate faults. There is an experiment to show the error masking capacity. We randomly inject faults to the output of each convolution layer at a certain ratio, and get an accuracy-ratio curve. We compare the results of zero injection and 128 positive perturbation injection on the 8-bit fixed-points like Figure 8 shows. The x-axis refers to the injection ratio and the y-axis refers to the accuracy. For positive perturbation injections, when the injection ratio reaches 1.5×10^{-4}, the accuracy of ResNet-20 has dropped below 90%. The accuracies of MobileNetV2 and GoogLeNet are even as low as 70%. For zero injection, before the fault ratio reaches 3×10^{-2}, the accuracy of ResNet-20 keeps above 90%.

B. Analysis of Error Detecting Capacity

In this section, we will discuss the capacity of our error detecting design for SEU. SEU is the most common soft error mode for FPGA implementation. As we have introduced some additional circuits in PE arrays, we will analyze the effectiveness of the proposed design by introducing SEUs in various circuit parts, including these additional circuits.

- SEU in PE Array: When the SEU occurs in the PE array, it is just the error mode we have anticipated. The proposed design will correctly execute traversal checking and locate the faulty PE.
- SEU in Pre-Defined Check Data Buffer: Since only one upset is considered, there will be no error in the PE, it will not affect the correctness of the comparator's results.
- SEU in Comparator: If the SEU occurs in the comparator, the outputs of the comparator will contain a great deal of error detected marks, which is different from the phenomenon of SEU in PE array. In the worst case, we can just regard such incorrect detecting as a False Alarm.
- SEU in Multiplexers: When the SEU occurs in these multiplexers, one of the PE will get incorrect check inputs, and the comparator will locate this PE as well.

In conclusion, our proposed error detecting design will not produce Missing Alarms for the SEU mode. While there will be possibilities of False Alarm when SEU occurs in the multiplexers and the comparators. Since the pages are limited, we do not analyze MBU here while it also has an extremely low probability of causing Missing Alarms.

C. Overhead

We implement our proposed PE error detecting design with RTL codes and in this section, we will provide some overhead results by EDA synthesizing.

In our synthesis of PE arrays, we use Xilinx Vivado 2018.2 tool and choose the board of Xilinx ZCU102 as the target device. We mainly list the consumption of LUT and FF resources for various parallelism parameters of PE arrays by Vivado synthesizing, as shown in Table III.

From Table III, we can see that the overhead of LUT is below 17% from parallelism (N, C, H, W)=(8,8,1,4) to (16,16,1,8), while the overhead of FF is below 9%. The overhead of both resources are much lower than that of traditional module redundancy schemes.

TABLE III
Overhead of the Proposed Error Detecting Design

Resources		LUT	FF
N=8,C=8 H=1,W=4	Conventional	12925	10629
	Proposed	14959	11497
	Overhead	15.74%	8.17%
N=12,C=12 H=1,W=4	Conventional	23889	22037
	Proposed	27745	23513
	Overhead	16.14%	6.70%
N=16,C=16 H=1,W=8	Conventional	73950	66821
	Proposed	86435	70541
	Overhead	16.88%	5.57%

VI. Conclusions

In this paper, we explore the influence of SEU on FPGA for neural network inference. From our results, SEUs occurring in PEs are more critical than in RAMs. Considering 2^{n-1} positive perturbation, if the FPGA accelerator has one PE failure in around 6000 PEs, the accuracy of GoogLeNet and MobileNetV2 would decline over 20%. So we must pay attention to SEU in PEs in practical applications.

To solve this problem, we propose a fault-tolerant design to detect and mitigate SEUs. We utilize the free cycles of neural network accelerators to reduce the overhead. The overhead of LUTs and FFs is below 17% and 9% respectively in our proposed design, which is much smaller than Xilinx TMR schemes. Besides, we can just traverse the higher bits and use zero settings to efficiently tolerance the errors temporarily.

Acknowledgments

The work of Y. Wang and H. Yang was supported in part by a 973 project and the National Natural Science Foundation of China under Grant 61532017. X. Chen's work was supported by Beijing Academy of Artificial Intelligence (BAAI).

References

[1] C. Z. et al., “Optimizing fpga-based accelerator design for deep convolutional neural networks,” in *FPGA*. ACM, 2015, pp. 161–170.

[2] J. Q. et al., “Going deeper with embedded fpga platform for convolutional neural network,” in *FPGA*. ACM, 2016, pp. 26–35.

[3] D. S. et al., “NASA’s approach to commercial cargo and crew transportation,” *Acta Astronautica*, vol. 63, no. 1, pp. 192–197, 2008.

[4] D. Hu, “Cots components radiation test activity and results at mssl,” in *IEEE Radiation Effects Data Workshop (REDW)*, 2016, pp. 1–7.

[5] F. L. K. et al., “On the optimal design of triple modular redundancy logic for sram-based fpgas,” in *DATE*. IEEE Computer Society, 2005, pp. 1290–1295.

[6] C. B. et al., “Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas,” in *DFT*. IEEE, 2007, pp. 87–95.

[7] Z. Z. et al., “Fine-grained module-based error recovery in fpga-based tmr systems,” *TRETS*, vol. 11, no. 1, p. 4, 2018.

[8] Y. H. et al., “Robust fpga resynthesis based on fault-tolerant boolean matching,” in *ICCAD*. IEEE, 2008, pp. 706–713.

[9] J.-Y. L. et al., “In-place decomposition for robustness in fpga,” in *ICCAD*. IEEE, 2010, pp. 143–148.

[10] J. S. et al., “In-place lut polarity inversion to mitigate soft errors for fpgas,” in *DFT*. IEEE, 2016, pp. 81–86.

[11] F. B. et al., “Comparative analysis of inference errors in a neural network implemented in sram-based fpga induced by neutron irradiation and fault injection methods,” in *SBCCI*. IEEE, 2018, pp. 1–6.

[12] F. L. et al., “Selective hardening for neural networks in fpgas,” *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2019.

[13] A. P. A. et al., “The effect of weight errors on neural networks,” in *CCWC*. IEEE, 2018, pp. 190–196.

[14] Y. I. et al., “Improving the robustness of a softcore processor against seus by using tmr and partial reconfiguration,” in *FCCM*. IEEE, 2010, pp. 47–54.

[15] Y. X. et al., “Dnnvm: End-to-end compiler leveraging heterogeneous optimizations on fpga-based cnn accelerators,” *arXiv preprint:1902.07463*, 2019.

[16] K. G. et al., “Angel-eye: A complete design flow for mapping cnn onto embedded fpga,” *TCAD*, vol. 37, no. 1, pp. 35–47, 2018.

[17] S. H. et al., “Ese: Efficient speech recognition engine with sparse lstm on fpga,” in *FPGA*. ACM, 2017, pp. 75–84.

[18] M. S. et al., “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *CVPR*, 2018, pp. 4510–4520.

[19] K. H. et al., “Deep residual learning for image recognition,” in *CVPR*, 2016.

[20] C. S. et al., “Going deeper with convolutions,” in *CVPR*, 2015, pp. 1–9.

[21] A. K. et al., “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.

[22] J. Y. et al., “Instruction driven cross-layer cnn accelerator for fast detection on fpga,” *TRETS*, vol. 11, no. 3, p. 22, 2018.

[23] Xilinx, “User guide: 7 series fpgas configurable logic block,” 2016, https://xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf.