FSA: An Efficient Fault-tolerant Systolic Array-based DNN Accelerator Architecture

Yingnan Zhao*, Ke Wang†, and Ahmed Louri*
*Department of Electrical and Computer Engineering, George Washington University, Washington, D.C.
†Department of Electrical and Computer Engineering, University of North Carolina at Charlotte, Charlotte, NC
Email: {yzhao96, louri}@gwu.edu, ke.wang@uncc.edu

***Abstract*—With the advent of Deep Neural Network (DNN) accelerators, permanent faults are increasingly becoming a serious challenge for DNN hardware accelerator, as they can severely degrade DNN inference accuracy. The State-of-the-art works address this issue by adding homogeneous redundant Processing Elements (PEs) to the DNN accelerator's central computing array, or bypassing faulty PEs directly. However, such designs induce inference loss, extra hardware cost, and performance overhead. Moreover, current designs are able to only deal with a limited number of faults due to costs. In this paper, we propose FSA, a Fault-tolerant Systolic Array-based DNN accelerator with the goal of maintaining DNN inference accuracy in the presence of permanent faults. The key feature of the proposed FSA is a unified re-computing module (RCM) that dynamically recalculates the required DNN computations that are supposed to be accomplished by faulty PEs with minimal latency and power consumption. Simulation results show that the proposed FSA reduces inference accuracy loss by 46%, improves execution time by 23%, and reduces energy consumption by 35% on average, as compared to existing designs.**

***Index Terms*—Fault-tolerant, systolic array, DNN accelerator**

I. Introduction

Permanent faults in the systolic array-based Deep Neural Network (DNN) accelerators [1]–[10] have a significant negative impact on DNN inference accuracy [11]. Prior works [12]–[28] have proposed both software and hardware solutions to address permanent faults in the central computing array (CA) of the DNN accelerator. Some of the existing software-based solutions [12], [13] propose to retrain the entire DNN model so that weight matrices will be updated for each DNN layer. Others [11], [12], [29] have proposed to avoid using the defective Processing Elements (PEs) for critical computations. However, these designs cannot cover all cases as the retrained DNN models and new mapping strategies are unlikely to converge with added constraints (e.g., irregular dataflow to avoid multiple faulty PEs). Hardware-based solutions implement redundant hardware components, including multiply-accumulate (MAC) units, PEs, and links, to avoid the negative impact of permanent faults [14]–[25], [27], [28], [30]–[33]. However, these techniques inevitably induce significant cost, excessive power consumption and latency.

To this end, we propose FSA, a fault-tolerant systolic array-based DNN accelerator design to address permanent faults. The objective of the proposed FSA architecture is to maintain the DNN inference accuracy in the presence of permanent faults with minimal overheads. The essence of the proposed FSA is a unified re-computing module (RCM) that re-computes the required computations by the faulty PEs of the CA with reduced latency and improved energy efficiency as compared to prior work [17]. The major contributions of this work are summarized as follows:

- **Reduced DNN Inference Accuracy Loss:** We propose a unified RCM that re-computes the required computations that are supposed to be done by faulty PEs. Unlike previous designs [15]–[17] that only address a limited number of faulty PEs, the proposed RCM design is able to reduce DNN inference accuracy loss regardless of the number or locations of the faulty PEs.
- **Improved Performance:** The proposed RCM works in parallel with the CA and provides the required DNN computations by faulty PEs with minimal latency and power consumption.
- **Performance Evaluation**: We evaluate the performance of the proposed FSA architecture with the ImageNet dataset [34], using the configurations of AlexNet and ResNet-50 [35], [36]. Simulation results show that the proposed FSA reduces inference accuracy loss by 46%, improves execution time by 23%, and reduces energy consumption by 35% on average, compared to existing fault-tolerant DNN accelerator designs [11], [15]–[18].

II. FSA Architecture Design

A. FSA Design Overview

Fig. 1 shows the overall architecture of the proposed FSA design that comprises of a systolic-based CA, on-chip buffers, and an RCM. The systolic-based CA consists of 256×256 PEs and deploys multiple dataflows: Output Stationary (OS), Weight Stationary (WS), and Input Stationary (IS). As shown in Fig. 2, the choice of dataflow reveals the reuse type of data operands (input, weight, or output) over space and time inside spatial accelerators. During the DNN inference phase, the inputs (activation and weight) are pre-loaded from the off-chip memory to the input buffers before being streamed into the PE array of the CA. The input buffer is constructed with a number of First-In-First-Out buffers (FIFO), each of which is assigned to a row or column of the CA. The input data stored in FIFO buffers are organized as the systolic array. Fig 3 shows the architecture of a conventional PE and it includes a multiplier–accumulator (MAC) and a register for storing data

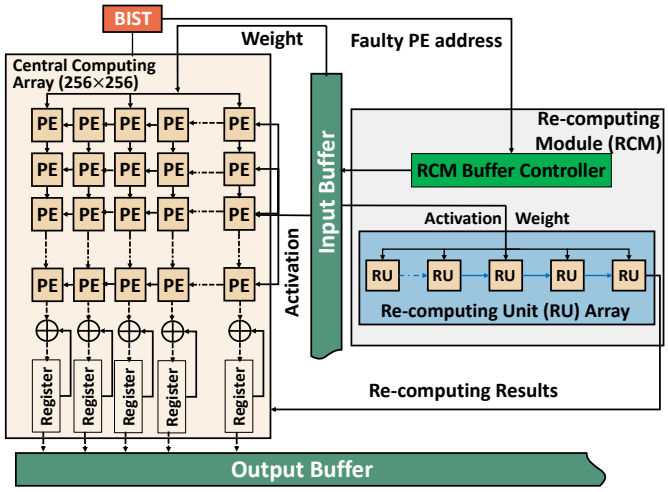

Fig. 1. The architecture of Fault-tolerant Systolic Array-based DNN Accelerators (FSA). PE: Processing Element, RU: Re-computing Unit

(activation, weight, and partial sum) locally. Based on the applied dataflow to the CA, PEs have to store different types of data in the local register. For the OS dataflow, PEs in the systolic array-based architecture receive data (activation and weight) from previous PEs, store them in the register file in the current cycle, and transmit them to the local MAC units and downstream PEs in the next cycle, simultaneously. The partial sum is stored and accumulated inside each PE and streamed to the output buffer until the calculation of the entire computing array is completed. For the IS and WS dataflows, input and weight matrices are pre-loaded to each PE's register for storing locally before performing required computations, respectively. Partial sums are streamed through PEs in the same column to obtain the desired accumulations. The proposed FSA design utilizes a unified RCM module to maintain the DNN inference accuracy in the presence of permanent faults. After the system detects faulty PEs, the partial sum of those faulty PEs are set to zero directly for the rest of the computation, while the working status of the remaining PEs are set to normal to ensure data synchronization and uninterrupted dataflow inside the CA. Faulty PEs whose outputs were set to zeros will be replaced by the correct results calculated by the RCM (Sec. II-B) so that the final DNN inference results is guaranteed. Note that the key feature of the proposed FSA design is to maintain DNN inference accuracy after permanent faults are detected.

B. Proposed Re-computing Module (RCM) Design

The proposed RCM module recomputes the product of activation and weight matrices originally mapped to the faulty PEs. The proposed RCM consists of an RCM Buffer Controller and Re-computing Units (RUs) as described next.

1) RCM Buffer Controller: We assume the Built-In-Self-Test (BIST) [37]–[39] is used to detect the faulty PEs inside the CA and store the location in the fault detection table located in the BIST module. The RCM buffer controller receives the address of faulty PEs from the BIST module including X and Y coordinates. As discussed in Sec.II-A, the input buffer consists of a number of FIFOs, each of which is mapped to a row or column of the CA. Therefore, using the coordinates of the faulty PE, the RCM buffer controller can easily locate the corresponding FIFOs and directly read the required activations and weights. Unlike previous designs [15]–[17] in which the required activation and weight data have to traverse the entire CA before being used by the redundant PEs, the proposed design eliminates excessive data transmission, thus reducing recalculation latency.

2) Re-computing Unit (RU): The objective of the proposed Re-computing Unit (RU) is to recalculate the partial sum required by a given faulty PE. The architecture of the proposed RU is shown in Fig. 4. Each RU integrates one MAC unit (an adder and a multiplier), one local register, and a set of MUX/DEMUX. The multiplier of the MAC unit calculates the product of weight and activation and forwards the result to the adder, and the adder then calculates the partial sum. The register is used to store the partial sum for accumulation locally and transmit the partial sum to the registers of CA to replace the supposed result of a faulty PE. To handle more than one faulty PE in FSA, the RU can be extended to an RU array. The set of MUX and DEMUX are used to connect the current RU to adjacent RUs and select links to output the local partial sum. In this case, the local register receives the partial sum from upstream RUs, and transmits it to downstream RUs and then to the SA's registers. The proposed design can also tolerate faulty RUs in the RCM by simply deactivating and bypassing the faulty RUs in the RU-array. The related deactivating and bypassing logic is not shown for clarity purposes.

III. FSA Operation

As mentioned above, we assume that the detection of faulty PE is done through a Built-In-Self-Test (BIST) strategy and is beyond the scope of this paper. During the DNN inference, the activation and weight are pre-loaded to the input buffer as mentioned in Sec. II-A. Based on the locations of faulty PEs and the dataflow deployed inside the SA, the RCM re-calculates the computation that is supposed to be done in the faulty PEs. For this, the RCM buffer controller fetches the corresponding activation and weight values from the input buffer (FIFO) and passes them onto the RU. The RU, in turn, performs the needed computation (the partial sum) and stores it in the local register. When the recalculation is complete, the RU configures the MUX-DEMUX circuitry so that the calculated correct partial sum is sent to the registers of CA for further overwrite the incorrect results. The RU continues to receive activation and weight values for recalculating the required computation of the next faulty PE. To address the situation when they are several faulty PEs, the RU can be extended to an RU array and RUs of the RU array can perform the required recalculations and transmit correct results concurrently.

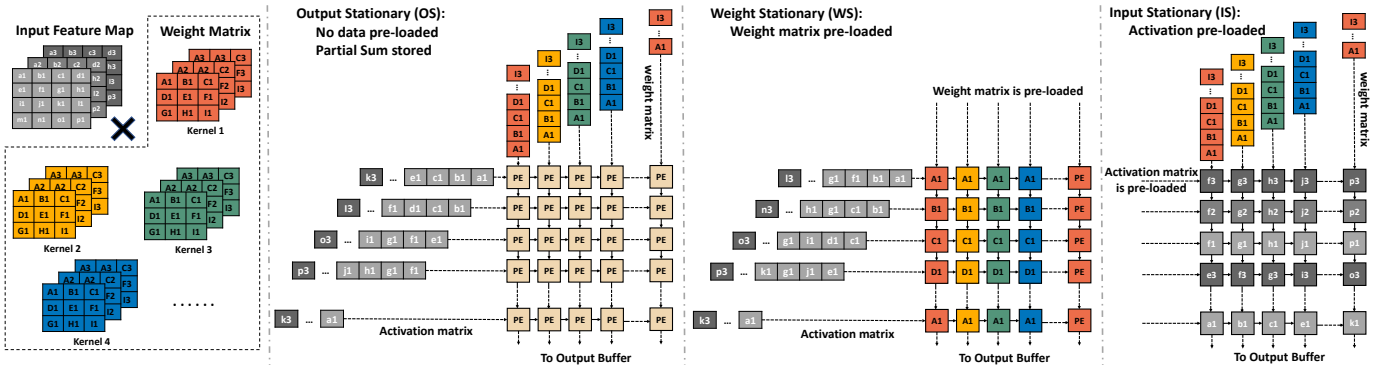

Fig. 2. Multiple dataflows applied to the systolic-based central computing array: (a) Output Stationary (OS): partial sums are accumulated and stored in each PE locally, (b) Weight Stationary (WS): weights are pre-loaded to each PE and stored locally, (c) Input Stationary (IS): inputs are pre-loaded to each PE and stored locally.

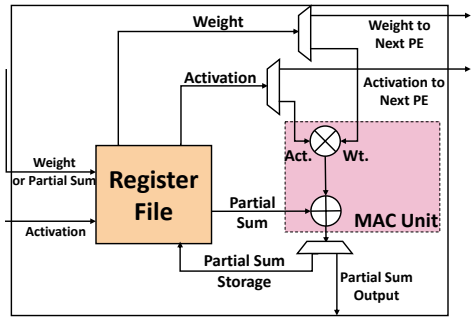

Fig. 3. The architecture of conventional Processing Element (PE) located in the central computing array. For a faulty PE, the MAC unit (shown in the red box) is eliminated from the rest of the computation.

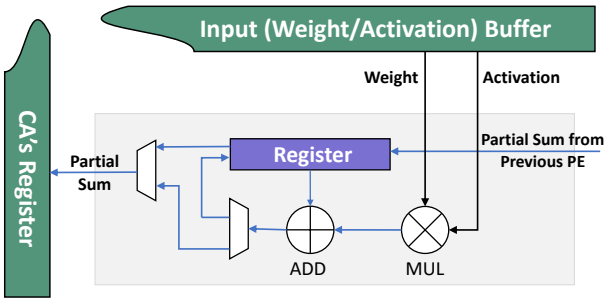

Fig. 4. The architecture of Re-computing Unit (RU) located in the RCM. The register is used to receive the data from previous RUs. The set of MUX-DEMUXes is used to select the output direction (register or downstream RU) of the partial sum

A. Output Stationary (OS) Workflow

Fig. 5 shows an example using an RU array with n RUs to address K faulty PEs in the $N \times N$ central computing array (in this example, $n = 2$, $K = 4$, $N = 4$, and the dataflow is OS). As shown in Fig. 5, the faulty PEs 0˜3 are responsible for calculating $\{a5-a8\}\times\{W5-W8\}$, $\{a9-a12\}\times\{W5-W8\}$, $\{a1-a4\}\times\{W9-W12\}$, and $\{a1-a4\}\times\{W13-W16\}$, respectively. Upon detection of the four faulty PEs, the partial sums of these PEs are set to zeros directly, and the rest of the central computing array calculates the matrix multiplication as usual. Meanwhile, two RUs, RU0 and RU1, are activated to calculate $\{a5-a8\}\times\{W5-W8\}$ and $\{a9-a12\}\times\{W5-W8\}$, respectively. From cycle *1* to *N*, the weights {W5-W8} are loaded from the on-chip buffer and sent to RU0 and RU1, as they use the same weights to calculate the partial sums. At the same time, $\{a5-a8\}$ and $\{a9-a12\}$ are streamed through the input port of RCM to RU0 and RU1, respectively. As mentioned in Sec. II-A, the on-chip buffers are constructed with FIFOs and each FIFO is assigned to each row and column of SA, respectively. Therefore, RCM buffer controller can read the required data directly from on-chip buffers based on the coordinate of faulty PEs. For the RU array, RU0 and RU1 calculate the partial sums simultaneously and store the partial sum in their local registers. The MUX-DEMUX inside each RU is inactive, and no data transmission between RUs, as shown in Fig. 5(b). At cycle *N*, after the RU0 and RU1 finish the calculations of $\{a5-a8\}\times\{W5-W8\}$ and $\{a9-a12\}\times\{W5-W8\}$, RUs change the configuration of MUX-DEMUX so that the calculated partial sums inside the local register is sent to the register of the CA through the RU array. At the same time, RU0 and RU1 continue to receive the data from the input port of RCM for recalculating the required computation for faulty PE2 and faulty PE3, as shown in Fig. 5(c). At cycle *2N*, after the RU array finishes the recalculation of faulty PE2 and PE3, only register and MUX are active so that the RU array can stream the partial sum from each RU's register to the register of the CA as shown in Fig. 5(d).

The RCM activates n RUs in FSA if K faulty PEs are detected. If K is smaller than n, K RUs will be activated to recalculate the required computation. In this case, the RCM calculation latency is overlapped with the execution time of the CA as the RUs can calculate all required partial sums concurrently. However, if K is larger than n, some RUs will have to calculate the results of multiple faulty PEs sequentially, which can lead to potential timing overheads. An $N \times N$ 2D systolic-based computing array consumes $3 \times N - 1$ cycles to calculate an $N \times N$ matrix multiplication. Since the computing array also needs N additional cycles to stream the partial

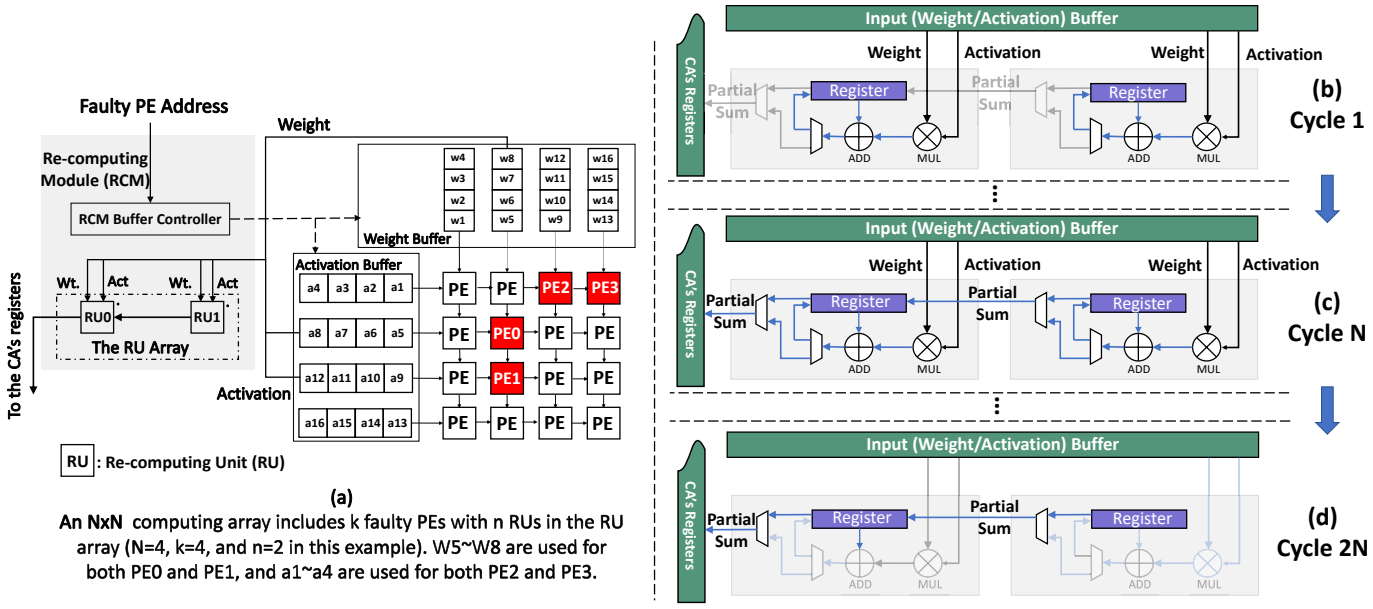

Fig. 5. An example of the working of FSA architecture for the OS dataflow: (a) An $N \times N$ systolic-based CA has K faulty PEs with n RUs (N=4, K=4, and n=2 in the example). (b) RU0 and RU1 receive data from the input port. (c) RU0 and RU1 finish the recalculation of PE0 and PE1, stream the results out, and receive input data of PE2 and PE3. (d) RU0 and RU1 finish the recalculation for all faulty PEs and stream results to registers of CA

sums to the register of the SA, the total execution time is $4 \times N - 1$. For RCM with n RUs, each RU requires N cycles for calculating a partial sum, and some RUs need to recalculate multiple partial sums sequentially when K is greater than n. Under this condition, the value of K/n will result in the different maximum number that FSA can tolerate without inducing extra timing overhead. If K can be divisible by n, all RUs in the RU array need to address the same number of faulty PEs, and the total execution time for recalculation is $N*(K/n)+n$. If K cannot be divisible by n, some RUs of the RU array need to address one more faulty PEs than others, and the total execution time is $N*(K/n)+K$ mod n. In this case, under-utilized RUs will be powered-off. Using these equations, the maximum number of faulty PEs K that FSA with n RUs can tolerate without inducing additional timing overhead can be calculated.

B. Weight Stationary (WS) Workflow and Input Stationary (IS) Workflow

FSA implements a similar mitigation workflow for both IS and WS dataflows to mitigate faulty PEs compared with the OS dataflow. Therefore, due to page limitations, we use the WS dataflow as an example to show how FSA reduces the inference loss with the presence of permanent faults. For simplicity, we use the same faulty condition in the previous Sec III-A, which includes a 4×4 CA with 2 RUs and 4 faulty PEs. For the WS dataflow, weights are pre-loaded from the on-chip buffer to the register of each PE and stored locally before performing the required computations. Partial sums are streamed and accumulated through PEs in the same column of the CA. As shown in Fig. 6, those PEs in red are faulty PEs and they store W6, W7, W12, and W16,

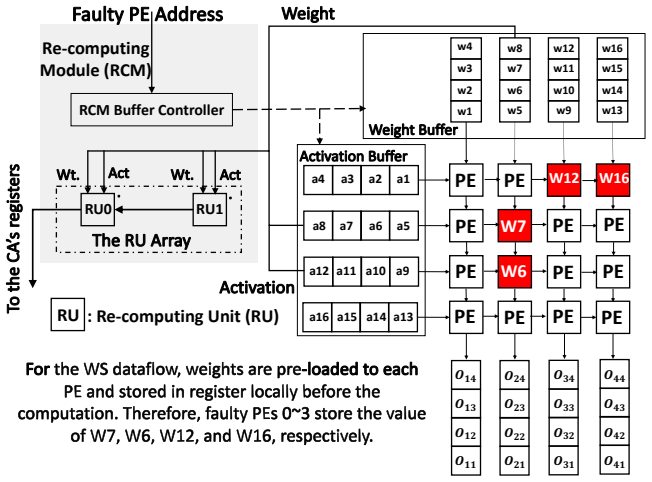

Fig. 6. An example of the working of FSA architecture for the WS dataflow: An $N \times N$ systolic-based CA has K faulty PEs with n RUs (N=4, K=4, and n=2 in the example). RU0 and RU1 receive data from the input port and perform the recalculation of PE0 and PE1, respectively. The recalculated partial sums are streamed from RUs to CA's registers for further accumulation. After RU0 and RU1 finishing all the recalculation of PE0 and PE1, they will move forward for PE2 and PE3, respectively.

respectively. In the WS dataflow, one faulty PE in a column of CA only impacts the value of local partial sum. Therefore, values of all outputs streamed through that faulty PE are incorrect. This is because when the output is streamed into the faulty PE, the added partial sum to the output is incorrect, which induces an incorrect output. For example, O_{21} equals $a_1 \times w_8 + a_5 \times w_7 + a_9 \times w_6 + a_{13} \times w_5$, as shown in Fig. 6. Because of two faulty PEs in the middle of column, the value of $a_5 \times w_7$ and $a_9 \times w_6$ are incorrect and the other two partial

sums are correct. Therefore, RUs only need to recalculate the incorrect partial sums instead of all partial sums compared with the OS dataflow. Similar to the OS dataflow, partial sums of four faulty PEs are set to 0 directly after the detection and other PEs of CA calculate the matrix multiplication as usual to keep the data synchronization.

At the beginning of computations, two RUs, RU0 and RU1, are activated to calculate the required partial sums, which are induced by faulty PEs. As mentioned before, the outputs are streamed in the vertical direction for related accumulation in the WS dataflow. As a result, RUs recompute the required computations induced by faulty PEs from the left column to the right of the CA. As shown in Fig. 6, RU0 recomputes the required computations induced by faulty PE0, and RU1 is responsible for the recalculations of PE1. For the input of RUs, the RCM buffer controller can fetch the required FIFOs directly based on the coordinate of faulty PEs, because each FIFO is assigned to each row of the CA. However, FSA works in a different way compared with the OS dataflow. For the WS dataflow, RCM only needs to read some specific weights from the FIFO instead of the entire one like the OS dataflow. Therefore, when the RCM buffer controller locates the specific FIFO based on the X-coordinate of faulty PEs, it needs to go through the related FIFO to search for the required weights based on the Y-coordinate of faulty PEs. As a result, the RCM buffer controller needs more cycles to locate the required weights compared with the OS dataflow. Assume that the size of each FIFO is N, the penalty for searching time is N$\times$(1+N)/2 cycles on average. After locating the specific weights (W6 and W7), they are streamed to RUs, respectively. At the same time, $\{a5 - a8\}$ and $\{a9 - a12\}$ are streamed through the input port of RCM to RU0 and RU1, respectively.

Obviously, RCM needs more time to locate the required weights for the WS compared with OS dataflow, which induces additional timing overhead. Therefore, the maximum number of faulty PEs that FSA can tolerate without inducing extra timing overhead is decreased compared with the OS dataflow. For the recalculation, RUs only calculate partial sums induced by faulty PEs in the WS dataflow. As shown in Fig. 6, RU0 calculates the value of a9$\times$W6 and RU1 calculates a5$\times$W7. After RUs finish the required computations, correct partial sums will be streamed to the downstream RUs instead of stored locally and they will be added to the final output through the CA's adders. When the recalculation of faulty PE0 and PE1 are completed, RU0 and RU1 will calculate the required computations induced by faulty PE2 and PE3, respectively.

IV. Evaluation And Analysis

A. Simulation Methodology

We evaluate the proposed FSA design using the SCALE-sim [40] simulator. We integrate CACTI [41] with SCALE-sim to evaluate power consumption. We compare the proposed FSA accelerator design to two previous techniques, namely Column Redundant (CR) [15]–[17], and Row Redundant (RR) [15]–[17]. For each technique, AlexNet and ResNet-50 [35], [36] are used as DNN models, and ImageNet is used as dataset [34]. For the proposed FSA, we implement 64 RUs (FSA-64) and 256 RUs (FSA-256), respectively.

1) Hardware Configuration of DNN Accelerators: We implement each technique (CR, RR, FSA-64, and FSA-256) in the SCALE-sim simulator. The CAs are of size 256×256 for all techniques. Each PE in the CA consists of a MAC unit and a local register. The PEs are connected with unidirection links. The horizontal links are 8-bit and used to transfer activation. The vertical links are 24-bit and used to transfer weight and partial sum. The on-chip input buffer size for each technique is 128 KB, which includes a 64 KB input feature buffer and a 64 KB weight buffer. The on-chip output buffer size is 192 KB for all designs. As mentioned before, we assume PE fault detection is done through a BIST strategy for all fault-tolerant techniques and is beyond the scope of this paper. For CR/RR, one column/row of homogeneous redundant PEs is added to the CA. Moreover, one 8-bit horizontal bypass link, one 24-bit vertical bypass link and switches are added to each PE, ensuring data synchronization and uninterrupted dataflow between the CA and redundant PEs. For the proposed FSA design, the RCM uses a data bus to multi-cast or unicast the activation and weight values from the input buffer to the RU array. We only evaluate the proposed FSA design with the OS dataflow because of the page limitations.

2) Fault Model: We use the Stuck-at fault [28] model to generate permanent faults. The value of the bit is either 0 or 1 if it has Stuck-at-0 or Stuck-at-1, respectively. Therefore, the result of faulty PEs will be randomly distributed. We execute the AlexNet and ResNet-50 DNN models 1,000 times for each experiment. For each execution, we randomly inject permanent faults in the CA and record the average inference accuracy, execution time, and energy consumption. Note that permanent faults may also exist in the proposed RU design, the RCM simply deactivates and bypasses the faulty RUs in the RU array. However, in this paper, we assume the RCM is fault-free in the simulation.

B. Inference Accuracy Analysis

We evaluate the reliability of the proposed FSA by measuring the DNN inference accuracy with the presence of permanent faults. The results of DNN inference accuracy are shown in Fig. 7. We varied the ratio of faulty PEs from 5% to 30%. We use the ImgaeNet [34], [42] as the dataset with the AlexNet [35] and ResNet-50 [36] as the DNN models to evaluate the proposed design, respectively. There are 1000 classes in the ImageNet dataset. For better comparison, we use the top-5 inference accuracy to evaluate both AlexNet and ResNet-50. The top-5 inference accuracy means that the DNN model selects five labels as the final outputs after computations. If the real label of the input image exists in the set of outputs from DNN models, the system will define the DNN model detects the input image successfully. If there is no faulty PEs, the top-5 DNN inference accuracy of AlexNet and ResNet-50 are 74% and 88% after a limited number of epochs, respectively. Without any protection, the inference accuracy of the baseline is around 0%. This is

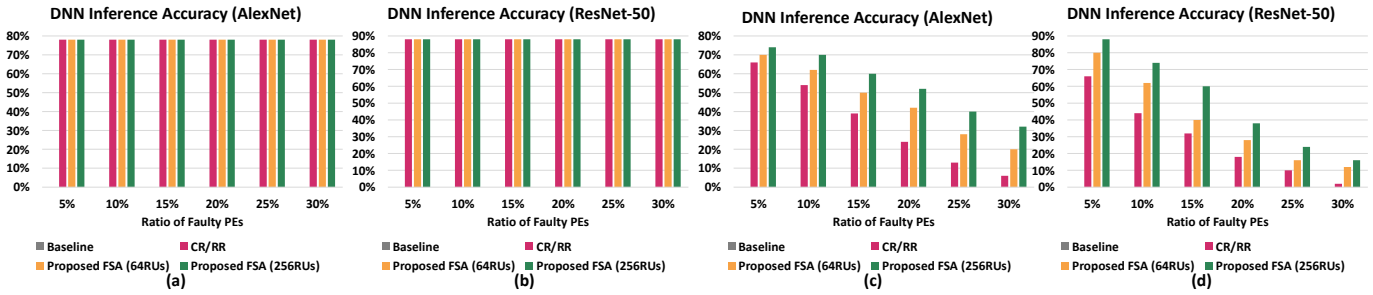

Fig. 7. DNN inference accuracy comparison. Inference Accuracy of the architecture without any fault-tolerant technique works as the baseline (0%). (a) AlexNet and (b) ResNet-50 without system run time constraint. (c) AlexNet and (d) ResNet-50 with system run time constraint.

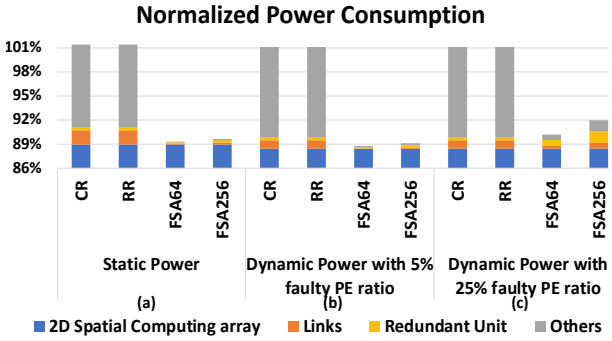

Fig. 8. Normalized power consumption breakdown. (a) Static power consumption, (b) Dynamic power consumption with 5% faulty PE Ratio, and (c) Dynamic power consumption with 15% faulty PE Ratio.

because the DNN model cannot figure out the input image with the presence of permanent faults and select one of the classes as the final output randomly. Therefore, the inference accuracy is around 0% (1/1000) without any protection. For the accelerators with protection, it should be noted that if there is no timing requirement (limitation of run time), all CR, RR, and the proposed FSA (with 64 RUs and 256 RUs) can achieve minimized inference accuracy loss, as all recalculation of faulty PEs can be completed, as shown in Fig. 7(a) and Fig 7(b). However, to show the benefit of using the proposed FSA design on both accuracy and system performance, we terminate the execution in this experiment as soon as the CA finishes its workload. It can be seen in Fig. 7(c) and Fig. 7(d) that the proposed FSA design (FSA with 64RUs and 256RUs) can achieve lower inference accuracy loss as compared to others. When the faulty PE rate increases to 15%, the DNN inference accuracy of the proposed FSA with 256 RUs is doubled, as compared to CR/RR for both the AlexNet and ResNet-50 DNN models. Among all ratios of faulty PEs for both AlexNet and ResNet-50, the proposed FSA with 256 RUs achieves lower accuracy loss than the one with 64 RUs. This is because FSA-256 has more RUs than FSA-64 so that it can address more faulty PEs with time constraints.

C. Performance Analysis

We evaluate the power consumption, execution time, and energy consumption for CR, RR, the proposed FSA-64 and the proposed FSA-256 with the OS dataflow. Different from the experiments in Sec. IV-B, all results are shown with the entire system (including the CA, redundant PEs, and the proposed RCM) completing their workloads.

Overall Static Power Consumption: Fig. 8(a) shows the breakdown of the overall static power consumption for each DNN accelerator design, and all values are normalized to the power consumption of the CA. The CR and RR integrate extra links between the redundant PEs and the CA which consume an additional 1.8% power overhead. CR and RR also use complex switches between adjacent PEs to ensure data synchronization, thus inducing 10.4% power overhead. Moreover, CR and RR add 256 redundant PEs to each column or row, which incurs 0.4% additional power consumption. The proposed FSA eliminates the additional links and switches added to the CA and only uses 8-bit links inside the RU array for data transmission. Moreover, with the same length, the power consumption of the 24-bit link is 2.9 times greater than that of the 8-bit link, and the 24-bit switch is 1.34 times than that of the 8-bit switch. Although the RCM uses an additional bus to transfer the data from the input buffer to the RU array, FSA still achieves the lowest power consumption compared to other techniques thanks to the reduced number of additional links and switches. As shown in Fig. 8, the power overhead of the proposed FSA with 256 RUs is 1.2% in total, which is 11% smaller than other techniques. It should be noted that if the BIST indicates there are no faulty PES, the entire RCM can be power-gated to save power. Such circuitry is not shown in Fig.2 for clarity.

Overall Dynamic Power Consumption: Fig. 8(b) and Fig. 8(c) show the breakdown of the overall dynamic power consumption for all techniques with 5% and 25% faulty PE ratios, respectively. The CR and RR do not have dynamic power management techniques (e.g., power-gating) for the redundant units, therefore, the redundant PEs are always activated regardless of the presence of faulty PEs. As a result, both CR and RR keep the same value of dynamic power consumption across all faulty PE ratios. Different from CR and RR techniques, redundant units (RUs) in the proposed FSA will be power-gated if no faulty PE is detected. As mentioned in Sec. III, FSA only enables a sufficient number of RUs depending on the number of faulty PEs. As a result,

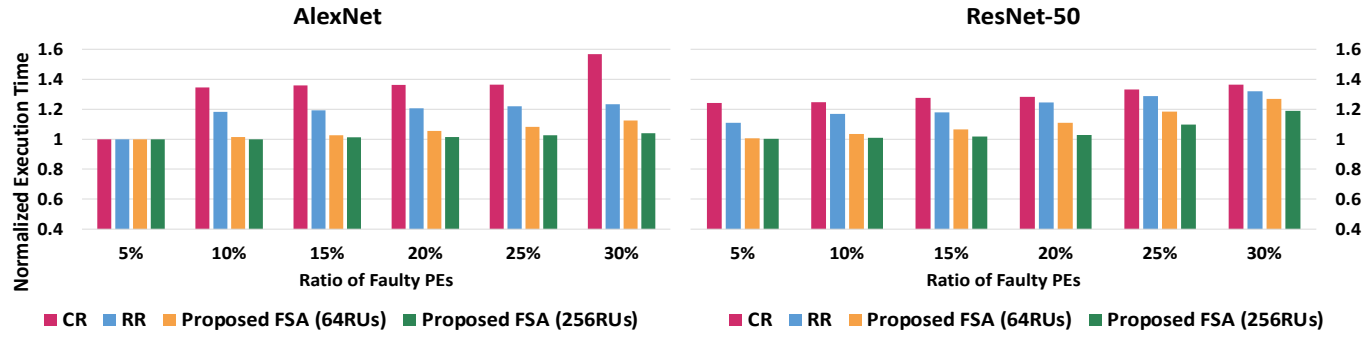

Fig. 9. Normalized execution time for two DNN models (AlexNet and ResNet-50) of four techniques (CR, RR, Proposed FSA-64 and Proposed FSA-256), with different ratios of faulty PEs ranging from 5% to 30%.

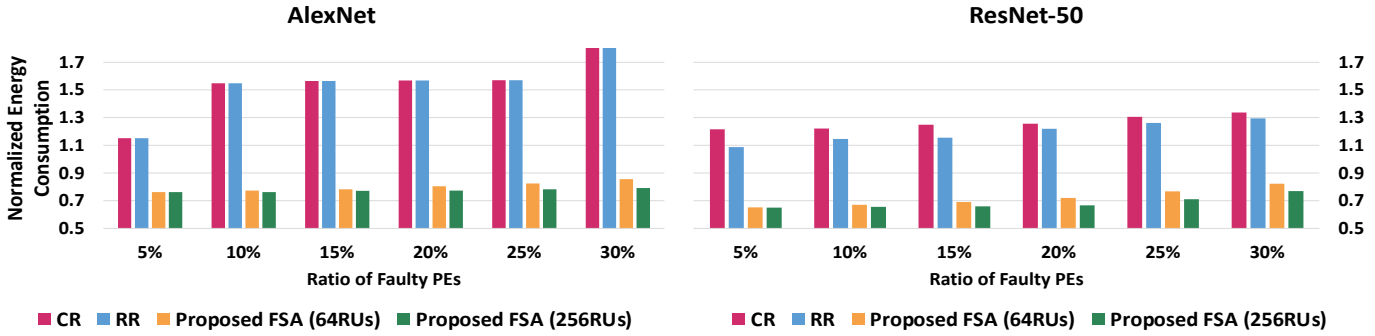

Fig. 10. Normalized energy consumption for two DNN models (AlexNet and ResNet-50) of four techniques (CR, RR, Proposed FSA-64 and Proposed FSA-256), with different ratios of faulty PEs ranging from 5% to 30%.

the dynamic power consumption of FSA will change based on the ratio of faulty PEs. Fig. 8(b) and Fig. 8(c) are examples of the dynamic power consumption of FSA in different ratios of faulty PEs. When the ratio of faulty PEs equals 5%, some of the under-utilized RUs inside FSA-256 will be power-gated to save power consumption as shown in Fig. 8(b). However, all RUs in the FSA-256 will be activated when the ratio of faulty PEs reaches 25%.

Execution Time: Fig. 9 shows the execution time for each DNN accelerator design. All values are normalized to the execution time of the CA without faults. It can be seen in Fig. 9, as the ratio of faulty PEs increases, the full execution time of the conventional CR and RR increase dramatically. The reason is the single-column/row redundant PEs can only handle a limited number of faulty PEs per row/column, and the row/column with the most faulty PEs will be the performance bottleneck. As we randomly select the faulty PEs, the locations of the faulty PEs are not evenly distributed for each row/column. Therefore the average execution time for CR and RR are significantly higher than the proposed FSA. As shown in Fig. 9, RR has a lower execution time than CR. For RR, the additional row of PEs is able to execute more input features at the same time. However, since the DNN models we use have limited kernels in each layer, DNN models may not benefit from the additional column of PEs in the CR design. Compared to prior works, the proposed FSA (FSA with 256 RUs) can tolerate more faulty PEs with a more significant number of RUs in RCM, achieving the lowest timing overhead.

Energy Consumption: We define the energy consumption as the product of the overall power consumption and the execution time. Fig. 10 shows the energy consumption for each fault-tolerant technique. All values are normalized to the power consumption of the CA without faulty PEs. As shown in Fig. 10, both FSA-64 and FSA-256 outperform CR and RR across all faulty PE ratios. The highest energy consumption reductions (93% and 76% using AlexNet and ResNet-50, respectively) are achieved when the faulty PE ratio equals 30%. The reasons are three-fold. First, the latency of the RCM of the proposed FSA is overlapped with the execution time of the CA, thus reducing execution time. Second, FSA integrates low-cost hardware components with reduced static power. Third, FSA deploys a dynamic power-gating strategy for under-utilized RUs and faulty PEs to achieve reduced dynamic power consumption.

Area Overhead: CR and RR add one column/row of 256 redundant PEs to the CA, which induce 0.9% additional area overhead. Moreover, CR and RR integrate additional links and switches for bypassing faulty PEs, transmitting data to the redundant PEs, and data synchronization. These additional links and switches consume 11.6% additional area overhead. Compared to CR and RR, the proposed FSA does not alter the CA with excessive links and switches, instead it adds an RCM unit for re-computation. The proposed RCM and the hardware components connecting RCM and CA consumes

3.1% additional area (FSA-256), which implies 75% smaller area overhead compared to CR and RR.

V. Conclusions

Permanent fault is a critical problem for Deep Neural Network (DNN) accelerators, as they usually lead to undesirable ramifications that severely impact DNN inference accuracy. Prior works address this issue by adding redundant PEs or bypassing faulty PE directly. However, such designs induce inference loss, extra hardware cost, and performance overhead. In this paper, we propose a fault-tolerant DNN accelerator design, named FSA, which integrates a systolic-based computing array and a re-computing module (RCM) for maintaining DNN inference accuracy with minimal latency and power consumption. The unified RCM consists of a number of re-computing units (RUs) that recalculate the required DNN computations mapped to faulty PEs found in the computing array regardless of the locations. Simulations using the configurations of AlexNet and ResNet-50 with the ImageNet dataset show that the proposed FSA reduces inference accuracy loss by 46%, improves execution time by 23%, and reduces energy consumption by 35% on average, as compared to existing designs.

Acknowledgement

The authors would like to thank the anonymous reviewers for their insightful comments. This work is partially supported by the National Science Foundation grants CCF-1702980, CCF-1901165, and CCF-1812495.

References

[1] Hsiang-Tsung Kung. Why systolic architectures? *Computer*, 1982.
[2] Norman P Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. of ISCA'17*, pages 1–12, 2017.
[3] Yu-Hsin Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE JSSC*, 2016.
[4] Hasan Genc et al. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *arXiv preprint arXiv:1911.09925*, 3:25, 2019.
[5] Ke Wang et al. Securenoc: A learning-enabled, high-performance, energy-efficient, and secure on-chip communication framework design. *IEEE TSUSC*, 7(3):709–723, 2021.
[6] Yuan Li et al. Ascend: A scalable and energy-efficient deep neural network accelerator with photonic interconnects. *IEEE TCAS-I*, 69(7):2730–2741, 2022.
[7] Jiajun Li et al. SGCNAX: A scalable graph convolutional neural network accelerator with workload balancing. *IEEE TPDS*, 33(11):2834–2845, 2022.
[8] Jiajun Li et al. Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks. In *Proc. of HPCA'21*, pages 775–788. IEEE, 2021.
[9] Ke Wang et al. AGAPE: anomaly detection with generative adversarial network for improved performance, energy, and security in manycore systems. In *Proc. of DATE'22*. IEEE, 2022.
[10] Ke Wang and Ahmed Louri. Cure: A high-performance, low-power, and reliable network-on-chip design using reinforcement learning. *IEEE TPDS*, 31(9):2125–2138, 2020.
[11] Jeff Jun Zhang et al. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *Proc. of VTS'18*, pages 1–6. IEEE, 2018.
[12] Jeff Jun Zhang et al. Fault-tolerant systolic array based accelerators for deep neural network execution. *IEEE Design & Test*, 2019.
[13] Jiacnao Deng et al. Retraining-based timing error mitigation for hardware neural networks. In *Proc. of DATE'15*. IEEE, 2015.
[14] André Flores dos Santos et al. Applying tmr in hardware accelerators generated by high-level synthesis design flow for mitigating multiple bit upsets in sram-based fpgas. In *Proc. of ARC'17*. Springer, 2017.
[15] Itsuo Takanami and Tadayoshi Horita. A built-in circuit for self-repairing mesh-connected processor arrays by direct spare replacement. In *Proc. of PRDC'12*, pages 96–104. IEEE, 2012.
[16] Itsuo Takanami and Masaru Fukushi. A built-in circuit for self-repairing mesh-connected processor arrays with spares on diagonal. In *Proc. of PRDC'17*, pages 110–117. IEEE, 2017.
[17] Itsuo Takanami et al. A neural algorithm for reconstructing mesh-connected processor arrays using single-track switches. In *Proc. of ICWSI'95*, pages 101–110. IEEE, 1995.
[18] Nahmsuk Oh et al. Error detection by duplicated instructions in super-scalar processors. *IEEE Trans. Reliab.*, 2002.
[19] Dawen Xu et al. Resilient neural network training for accelerators with computing errors. In *Proc. of ASAP*. IEEE, 2019.
[20] Ruochen Wang and Zhe Xu. A pedestrian and vehicle rapid identification model based on convolutional neural network. In *Proc. of ICIMCS'15*, pages 1–4, 2015.
[21] Ying Wang, Huawei Li, and Xiaowei Li. Frequency scheduling for resilient chip multi-processors operating at near threshold voltage. In *Proc. of DATE'16*, pages 1164–1167. IEEE, 2016.
[22] Li Li et al. Squeezing the last mhz for cnn acceleration on fpgas. In *Proc. of ITC-Asia'19*, pages 151–156. IEEE, 2019.
[23] Len Levine and Ware Meyers. Special feature: Semiconductor memory reliability with error detecting and correcting codes. *Computer*, 9(10):43–50, 1976.
[24] Yung-Chang Chang et al. On the design and analysis of fault tolerant noc architecture using spare routers. In *Proc. of ASP-DAC'11*, 2011.
[25] Wen-Chung Tsai et al. A fault-tolerant noc scheme using bidirectional channel. In *Proc. of DAC'11*, pages 918–923, 2011.
[26] Ke Wang et al. TSA-NoC: Learning-based threat detection and mitigation for secure network-on-chip architecture. *IEEE Micro*, 40(5):56–63, 2020.
[27] Brandon Reagen et al. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proc. of ISCA'16*, 2016.
[28] Behzad Salami et al. On the resilience of rtl nn accelerators: Fault characterization and mitigation. In *Proc. of SBAC-PAD'18*. IEEE, 2018.
[29] Muhammad Abdullah Hanif and Muhammad Shafique. Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping. *Philosophical Transactions of the Royal Society A*, 378(2164):20190164, 2020.
[30] Ke Wang et al. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *Proc. of DATE'19*, pages 1166–1171, 2019.
[31] Ke Wang et al. Intellinoc: A holistic design framework for energy-efficient and reliable on-chip communication for manycores. In *Proc. of ISCA'19*, 2019.
[32] Hao Zheng et al. Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures. In *Proc. of HPCA'21*, 2021.
[33] Hao Zheng et al. A versatile and flexible chiplet-based system design for heterogeneous manycore architectures. In *Proc. of DAC'20*, 2020.
[34] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 25, 2012.
[35] Md Zahangir Alom et al. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.
[36] Kaiming He et al. Deep residual learning for image recognition. In *Proc. of CVPR'16*, pages 770–778, 2016.
[37] Andreas Steininger and Christoph Scherrer. On the necessity of on-line-bist in safety-critical applications-a case-study. In *Digest of Papers. FTCS (Cat. No. 99CB36352)*, pages 208–215. IEEE, 1999.
[38] J.P. Hofmeister et al. Real-time bist detector for bga faults in field programmable gate arrays (fpgas). *Ridgetop Group, Inc. white paper*, 2006.
[39] Praveen Parvathala et al. Frits-a microprocessor functional bist method. In *Proc. of ITC'02*, pages 590–598. IEEE, 2002.
[40] Ananda Samajdar et al. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.
[41] Naveen Muralimanohar et al. Cacti 6.0: A tool to model large caches. *HP laboratories*, 27:28, 2009.
[42] Jia Deng et al. Imagenet: A large-scale hierarchical image database. In *Proc. of CVPR'09*. Ieee, 2009.