STRAIT: Self-Test and Self-Recovery for AI Accelerator

Hayoung Lee, *Graduate Student Member, IEEE*, Jihye Kim, Jongho Park, and Sungho Kang, *Senior Member, IEEE*

***Abstract*—As the demand for data-intensive analytics has increased with the rapid advance in artificial intelligence (AI), various AI accelerators have been proposed. However, as AI-based solutions have been adapted to applications requiring accuracy and reliability, the reliability of them has become a critical issue. For this reason, self-test and self-recovery for AI accelerator (STRAIT) is proposed in this article. It facilitates self-test, self-diagnosis, and self-recovery by utilizing the structural and operational characteristics of systolic array in AI accelerator. The proposed self-test is progressed using scan chains composed of functional paths and can achieve a 100% test coverage (for both stuck-at and transition-delay faults) with a small number of test patterns and reduced test power. The proposed self-diagnosis is progressed with the proposed self-test in real time and allows accurate fault localization with fault type analysis. The proposed self-recovery is progressed using efficient pruning for faulty processing elements with weight allocation, and the reliability of AI accelerators can drastically increase with negligible performance degradation. However, STRAIT can be implemented with a small area overhead.**

***Index Terms*—Artificial intelligence (AI) accelerator, built-in self-recovery (BISR), built-in self-test (BIST), fault diagnosis, fault recovery, scan test, systolic array.**

I. Introduction

With the introduction of big data and advances in computing power, artificial intelligence (AI)-related technologies have been rapidly developed. AI is widely used in diverse fields ranging from applications closely related to daily life, such as media processing and recommendation systems, to applications requiring accuracy and reliability, such as medical diagnosis, self-driving vehicles, security, and financial products [1], [2], [3], [4], [5]. However, as the main operations of AI are multiplication and convolution, they are time consuming at software level when a large amount of data is involved. Accordingly, hardware-based solutions have been developed such as AI accelerators [6], [7], [8], [9], [10]. It is possible because the main operations are composed of the iterations of simple multiplications and accumulations. Fig. 1 shows an example of a systolic array-based AI accelerator, which is a representative structure. As shown, the systolic array is composed of multiple processing elements (PEs), which are connected in two dimensions. Each PE is also composed of a multiply–accumulate (MAC) unit with multiple registers. For this reason, each PE progresses simple multiplication using weight and activation data, and the calculated data in each PE is accumulated while transferring the calculated data to the accumulator. Therefore, as the time-consuming problem at software level can be solved, the performance of AI can be maximized by using such hardware-based solution.

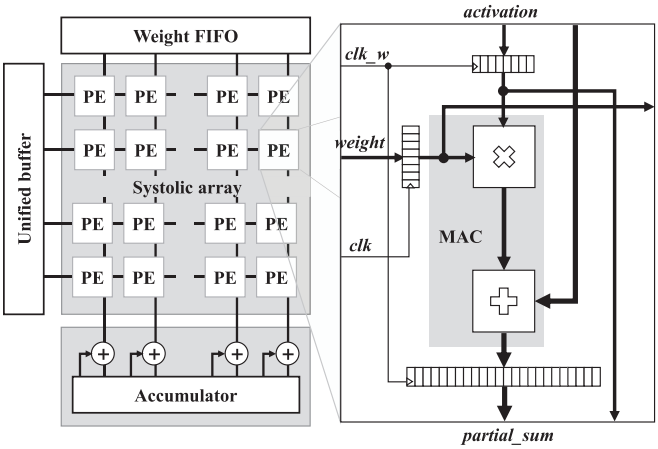

Fig. 1. Example of systolic array-based AI accelerator.

However, as the calculated data is accumulated, the effects of permanent faults can be also accumulated during operation if permanent faults occur in AI accelerator. If AI is applied on fault-tolerant applications such as neural network models, the accumulated effect by permanent faults can be slightly allowed [11], [12], [13]. On the other hand, if AI accelerators are applied on applications, which is directly related to safety, such as automotive and aviation, the permanent faults can cause serious consequences. Moreover, the systolic array is susceptible to manufacturing defects owing to densely integrated PEs [14]. In addition, the accuracy of an inference decreases significantly, even when the permeant fault rates of AI accelerators are low [11], [15]. For this reason, methods for testing and recovering the permanent faults have been required for reliable AI accelerators. Accordingly, several methods for testing and recovering the permanent faults

Manuscript received 23 August 2022; revised 18 November 2022; accepted 10 January 2023. Date of publication 13 January 2023; date of current version 22 August 2023. This work was supported by the MOTIE and KEIT (Design for Test of Intelligent Processors) under Grant 20012010. This article was recommended by Associate Editor G. D. Natale. *(Corresponding author: Sungho Kang.)*

Hayoung Lee, Jongho Park, and Sungho Kang are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, South Korea (e-mail: yseehy214@soc.yonsei.ac.kr; jongho0117@soc.yonsei.ac.kr; shkang@yonsei.ac.kr).

Jihye Kim is with the Foundry Business, Samsung Electronics, Hwaseoung 18448, South Korea (e-mail: jh152.kim@samsung.com).

Digital Object Identifier 10.1109/TCAD.2023.3236875

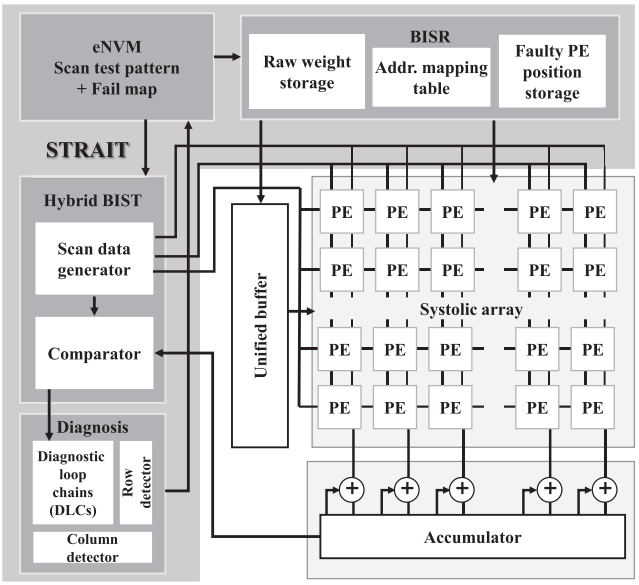

Fig. 2. Overall structure of STRAIT.

TABLE I
FUNCTIONALITY OF CONVENTIONAL METHODS

Architectures	[11]	[17]	[18]	[19]	[21]	[22]	[23]	[24]
Test	X	O	O	O	X	X	X	X
Diagnosis	X	X	O	X	X	X	X	X
Recovery	O	X	X	X	O	O	O	O

have been developed. However, no method has been proposed to support both test and recovery; all conventional methods focus on one or the other. Therefore, as two different methods for test and recovery need to be applied, additional overheads drastically increase.

In this article, self-test and self-recovery for AI accelerator (STRAIT) is proposed for realizing AI accelerators with high reliability. As shown in Fig. 2, STRAIT consists of three functions, namely, hybrid built-in self-test (BIST), diagnosis, and built-in self-recovery (BISR). The hybrid BIST module progresses self-test using scan chains composed of functional paths. The diagnosis module is simultaneously operated with the hybrid BIST module and progresses fault localization in real time. The BISR module progresses efficient pruning for faulty PEs with weight allocation. Scan test patterns are stored in embedded nonvolatile memory (eNVM), and the diagnostic results are also stored in eNVM after fault diagnosis processes. Also, the diagnostic results in eNVM are transferred to the BISR module for fault recovery. STRAIT significantly increases the robustness of AI accelerators against fault-induced errors beyond their inherent levels of fault tolerance. In addition, it is possible to secure the reliability of inferences by ensuring that the AI accelerator can perform everything from test to diagnosis and fault recovery on its own functions embedded in the chip. The main contributions of this article are as follows.

1) STRAIT is a unified solution, which can support self-test, self-diagnosis, and self-recovery for AI accelerator. No conventional method has been proposed to support both test and recovery as shown in Table I. Accordingly, the hardware overhead is reduced by sharing the logics for fault test and recovery in the systolic array.
2) In self-test of STRAIT, scan chains are constructed using functional paths of all registers. For this reason, the optimal test coverage can be achieved for both stuck-at (SA) and transition-delay (TD) faults with a small number of test patterns due to efficient PE-level scan testing. In addition, the hardware overhead can be highly reduced to construct a scan structure in the systolic array.
3) In self-diagnosis of STRAIT, fault localization can be progressed in real time. It is possible because diagnostic loop chains (DLCs) are used. Fault localization is simultaneously progressed with a scan test on the systolic array, additional time is not required.
4) In self-recovery of STRAIT, the main contributions are weight swapping and weight allocation. They are proposed based on pruning, which is the partial sum inputs are directly connected to the partial sum outputs with skipping the faulty MAC. However, if the pruned PE weight is a large value or the pruning of nonzero weights is accumulated, the possibility of error occurrence increases. To address the problem, weight swapping and weight allocation are used in STRAIT. In weight swapping, if a faulty PE and a PE with a zero weight exist in the same row or column, their weights are swapped. In weight allocation, if zero weights of the weight row can be assigned to all faulty PE locations in a faulty row of the systolic array, the weight row is allocated to insert into the faulty row of the systolic array. For this reason, a high recovery rate can be achieved without accuracy loss. In addition, most faulty recovery methods do not consider registers in PE. However, if faults occur in the registers, the effects of these faults are propagated to adjacent PEs. Therefore, it should be considered, and STRAIT supports fault recovery considering such registers.

II. BACKGROUND

A. Structural Test and Diagnosis Methods

The systolic array-based AI accelerator test can be progressed differently with the general system-on-a-chip (SoC) test [16]. It is because as each PE in the systolic array is composed of the same structure, all PEs can be tested with the same test patterns. For this reason, PE-level testing has been researched to test the systolic array. In [17] and [18], reconfigurable scan chain construction methods with PE grouping have been proposed. In this method, four PEs are grouped to form subarrays (SPEs), and scan chains are constructed in each SPE. It is because each PE receives data from PEs on its left and top and sends data to PEs on its right and bottom. After then, PEs at the same location in each SPE are tested simultaneously. However, whenever the target PEs are changed, scan chains should be reconstructed with different PE grouping. In addition, although it can achieve the reduction of test pattern counts compared to the conventional array-level testing, the hardware overhead drastically increases since additional hardware for scan chain construction and reconfiguration is

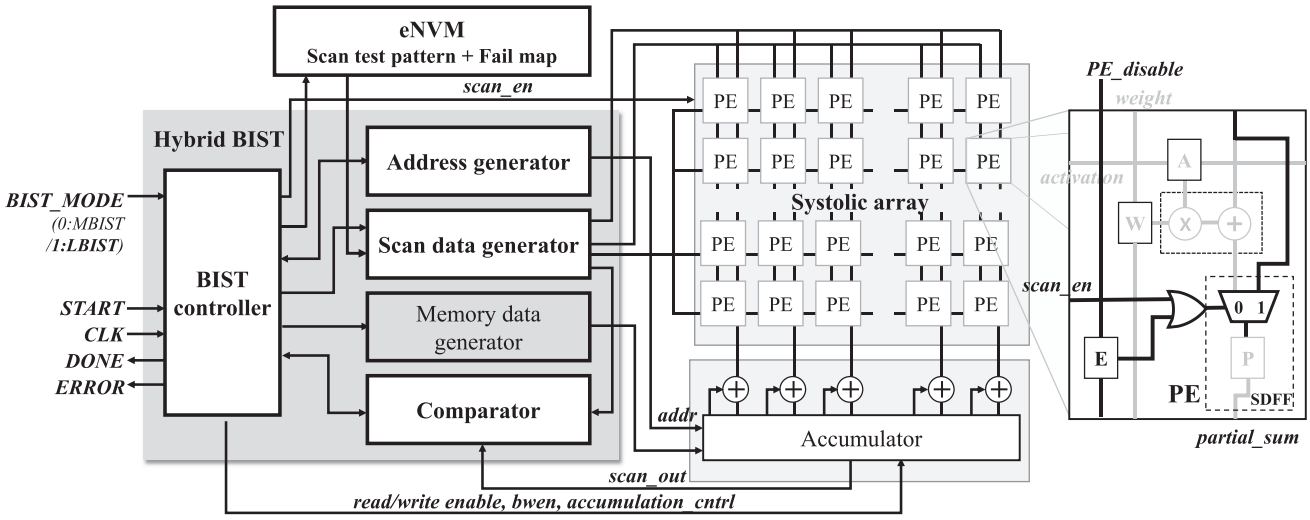

Fig. 3. Test structure with hybrid BIST.

inserted. On the other hand, the fault diagnosis method has been also proposed in [18]. In this method, the compressed outputs at target diagnosis level are stored and compared to one another. However, the post-processing time is required for fault localization, and the area and analysis time overheads for output storage and comparison increase with the increase in the diagnostic resolution.

In [19], the functional connections of the weight and activation registers are used to construct the scan structure. Only partial sum registers in each PE constitute new scan chains in the row direction. For this reason, the hardware overhead is reduced compared to [17] and [18]. In addition, since the same test values are applied on all PEs simultaneously, the weight and activation data shifted to adjacent PEs are identical without transition. For this reason, it was possible to reduce the scan shift power. However, the TD fault test coverage estimation is unclear. The TD fault test coverage is correct in [19] when only a row in the systolic array is considered. However, if the entire systolic array is considered, the TD fault test coverage can be different since the values in partial sum registers are continuously changed during the TD test. In addition, diagnosis cannot be progressed in this method.

B. Repair and Recovery Methods

In general, it is difficult to repair logic owing to structural diversity compared to memory, which has a uniform cell structure [20]. However, as the systolic array is a 2-D array of identical PE units, it has a uniform structure. For this reason, PE repair methods using redundant PE groups have been proposed. In [21], column-redundant PE groups are inserted. For this reason, if a faulty PE occurs, the faulty PE is substituted by a fault-free PE in other columns. However, as additional column-redundant PE groups are inserted, the hardware overhead has drastically increased. In addition, additional delay can be generated in functional paths. In [22], faulty PE units are repaired with both divided row- and column-redundant PE groups. However, the hardware overhead is large since additional redundant PE groups are inserted, and the repair rate is strongly affected by the number and location of faults. In [11] and [23], fault recovery methods using bypass paths have been proposed. When the bypass mode is activated, the partial sum inputs are directly connected to the partial sum outputs with skipping the faulty MAC. However, if the pruned PE weight is a large value or the pruning of nonzero weights is accumulated, the possibility of error occurrence increases. Although faulty recovery methods without accuracy loss have been also proposed by retraining after weight pruning, it is inefficient to retrain whenever a new fault occurs, if AI accelerators only support inference. In [24], a software method has been proposed that the faulty PEs with the critical faults are bypassed while leaving noncritical faults. It does not require additional hardware, but functional performance can deteriorate since the entire PE column is disabled even if there is one faulty PE. Moreover, even if all zero values are applied to the faulty columns or rows, the effects of defects are not masked perfectly.

III. Built-in Self-Test and Self-Diagnosis

A. Test Processes With Scan Chain Construction

To test the systolic array, functional paths are reused to construct scan chains. Similar to [19], all activation registers in the same row become a scan chain using functional paths. Likewise, all weight registers in the same column become a scan chain using functional paths. On the other hand, all partial sum registers in the same column also become a scan chain in STRAIT. Fig. 3 shows the test structure with hybrid BIST. As shown, to reuse partial sum registers for scan chain construction, an additional multiplexer controlled by *scan_en* signal is inserted in each PE. (An OR gate and an additional register in each PE are used for self-recovery, which is described in Section IV.) For this reason, if *scan_en* signal is set to 1, the input of the partial sum register is connected to the output of the upper partial sum register. Therefore, scan paths are constructed with partial sum registers in the same column, and they are used for test data shift. In this case, as each scan chain consists of flip-flops with the same bit location relative to each PE, the hardware overhead for scan chain construction is reduced.

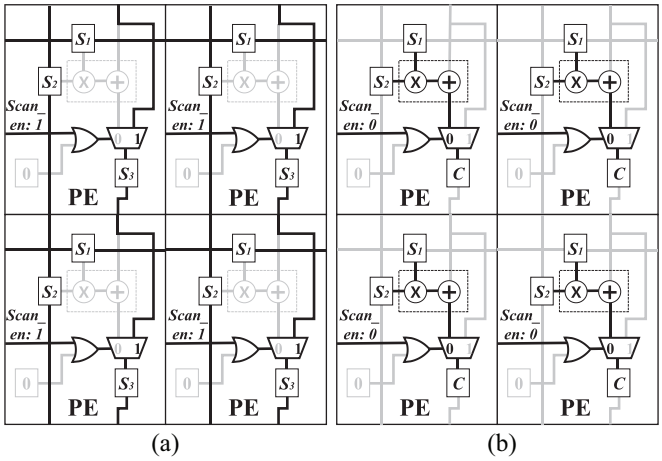

Fig. 4. Examples of SA test in STRAIT. (a) Scan shift processes and (b) scan capture processes.

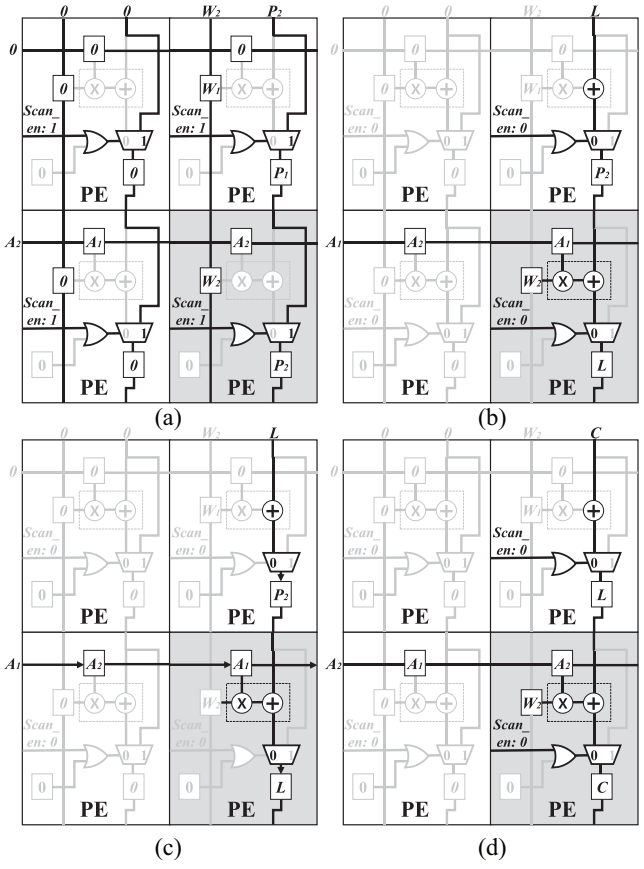

Fig. 5. Examples of TD test in STRAIT. (a) Scan shift processes, (b) launch clock application, (c) at-speed propagation, and (d) capture clock application.

Fig. 4 shows the examples of SA test processes using such constructed scan chains in STRAIT. As shown in Fig. 4(a), after *scan_en* signal is set to 1, all test data (S_1, S_2, and S_3) is inserted and shifted in each scan chain. In the case of SA test processes, as all PEs can be tested with the same test data, the same test data is inserted in each PE. If all test data insertions are over, capture processes are progressed after *scan_en* signal is set to 0, as shown in Fig. 4(b). After then, the capture data (C) is out with the next test data insertion. When the capture data analysis is progressed for SA fault detection, activation and weight scan data need not to be measured directly because the activation and weight operation results are already reflected in the data captured in the partial sum registers. Therefore, it is reasonable to measure the data captured in the partial sum registers. In addition, as the expected data captured in the partial sum registers is the same regardless of PEs, SA fault detection can be possible by simple comparisons [18], [19]. For this reason, the comparisons are simply progressed by the comparator in hybrid BIST. In this manner, all PEs are tested with the same test data, and the expected data after the test is the same regardless of PEs. For this reason, the shift power, which is weak in scan tests, can be drastically reduced. Furthermore, the number of required test data highly decreases with achieving the optimal test coverage since partial sum registers are constructed as scan chains. For example, only 12 test patterns, which are the result of automatic test pattern generation (ATPG), are required to achieve a 100% test coverage for SA faults in a 256 × 256 systolic array.

On the other hand, TD faults are also considered in STRAIT. In general, functional operations in a weight stationary-based systolic array begin after data in the weight registers is set, which is possible since the clock inserted into weight registers is independent of clocks of other registers. For this reason, the paths from the weight registers to the partial sum registers are not in the at-speed domain. In other words, the weight registers simply serve as a value setting for data propagation during the TD test, and the clock inserted into the weight registers is not applied during the capture procedure. Therefore, TD faults between activation and partial sum registers are mainly considered in the TD test, and clocks applied to the registers should be toggled with at-speed for multiple cycles during the capture procedure. The first cycle is known as the launch cycle, and the second cycle is known as the capture cycle. Additionally, the activation data is inserted from the left PE, and the partial sum data is inserted from the upper PE. For this reason, three PEs should be considered simultaneously: 1) the target PE; 2) the upper PE; and 3) the left PE. Accordingly, ATPG is progressed with a target PE, the corresponding upper PE, and the corresponding left PE. During ATPG, the clock applied to the weight registers is declared as a scan clock, but a constraint is given so as not to be toggled in the capture procedure. It is because the weight registers are not in the at-speed domain. In addition, target faults are limited to the at-speed domain in the target PE. After then, the same test patterns generated by ATPG are inserted into the entire systolic array. However, in the TD test of STRAIT, four PEs are grouped and PEs at the same location in each PE group are tested simultaneously to improve ATPG efficiency and decrease the toggle ratio. For this reason, test patterns for the TD test are inserted in only scan chains passing through the target PEs, and zero values are inserted in other scan chains to minimize the toggle ratio for shift and capture operations. The TD test of STRAIT can be performed on the entire systolic array if the same test processes are progressed with applying all TD patterns and changing the target PEs.

Fig. 5 shows the examples of the TD test in STRAIT. As shown in Fig. 5(a), in scan shift processes, test data is inserted in only scan chains passing through the target PEs, and zero

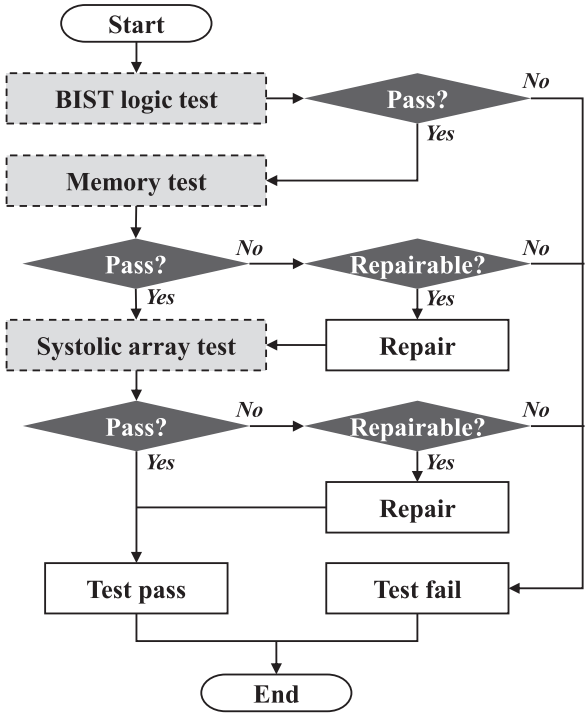

Fig. 6. Overall test scenario in STRAIT.

values are inserted in other scan chains. In this case, all values inserted in each register are propagated through functional paths for enough time before the launch cycle. After then, the first capture process is progressed after *scan_en* signal is set to 0. In the first capture process, as soon as the launch clock is applied, the first captured data (L) and activation value (A_1) in the target PE are immediately launched with launching the partial sum value (P_2) in the upper PE, as shown in Fig. 5(b). From this point on, the transition data must be propagated to the partial sum register in the target PE within the functional clock period as shown in Fig. 5(c). After the functional clock period, the capture clock is applied and the final captured data (C) is stored in the partial sum register of the target PE, as shown in Fig. 5(d). After then, the final captured data is out with the next test data insertion. If data propagation is not properly progressed during the functional clock period, faulty information is reflected in the second capture data. When the capture data is analyzed for TD fault detection, the comparisons are simply progressed, which is similar to SA fault detection. The TD test can be performed on the entire systolic array if the same processes are applied with changing the target PEs. In the case of the TD test, the number of required test data is reduced with the optimal test coverage achievement. For example, only 18 test patterns, which are the result of ATPG, with four iterations are needed to achieve a 100% TD fault test coverage in a 256 $\times$ 256 systolic array.

B. Hybrid BIST

For SoC chips with embedded memories, the memory test using memory BIST (MBIST) is essential. For this reason, systolic array-based AI accelerators with embedded memories, such as unified buffer and accumulator, are also tested using such MBIST. The basic MBIST consists of the MBIST controller, address and data generators, and comparators. When the memory test begins, the required memory control signals are generated in MBIST controller by following the built-in or user-defined test algorithm. Address and data generators then generate necessary memory addresses and test data. Thereafter, the generated signals are sent to the memories and comparators. Sequentially, data comparisons are progressed in comparators while all required memory write/read operations are progressed. Finally, the test results are sent to the MBIST controller for determining the final pass/fail verdict.

On the other hand, logic BIST (LBIST) is widely utilized for logic test. LBIST is appropriate for various logic structure since test patterns are generated by pseudo-random pattern generator and test response data is compared by multiple-input signature registers, but it is difficult to achieve a high test coverage since logic structures are complex and vary. However, as the systolic array has a uniform structure, a high test coverage can be achieved with LBIST. In addition, as the number of test patterns required for systolic array test is extremely small, the test coverage can more increase with deterministic test patterns. To implement LBIST for systolic array test, LBIST controller, address and scan data generators, comparators, and eNVM are required in STRAIT. The LBIST controller is required to control LBIST submodules and drive systolic array and accumulator block according to the scan procedure. The scan data generator is required to provide scan-in data and the expected scan-out data to the scan input and comparators, respectively. The address generator is required to apply appropriate read/write memory addresses in the internal memory units of accumulator block, since scan-out data from the systolic array is temporarily stored in accumulator block. Comparators are required to compare the scan-out data stored in the accumulator with the expected scan-out data obtained from the scan data generator and send the test results to the LBIST controller for determining the final pass/fail verdict. eNVM is required to store deterministic test patterns for the systolic array test. In this case, the deterministic test pattern counts are extremely low. Also, as scan-in data is shared and the expected scan-out data has the same value, the number of data bits to be stored for each test pattern is small. Consequently, even if all deterministic patterns are stored in eNVM, the hardware overhead by test patterns is negligible.

Owing to the similarities between general MBIST and the proposed LBIST, most hardware can be shared. For this reason, hybrid BIST is proposed in this article, which is capable of two operation modes: 1) MBIST and 2) LBIST. As shown in Fig. 3, scan tests on the systolic array can be performed by updating the functions of several control logics to execute the scan procedure and by adding a scan data generator that applies scan input and output data. Fig. 6 represents the overall test scenario with the proposed hybrid BIST. As shown, scan tests on hybrid BIST are preferentially performed, which are general test procedures in SoC. It is because the test results of memories and the systolic array are reliable with only fault-free hybrid BIST. In this case, if any fault is detected in hybrid BIST, the chip should be discarded since recovery is difficult for such logic circuits. After then, the MBIST

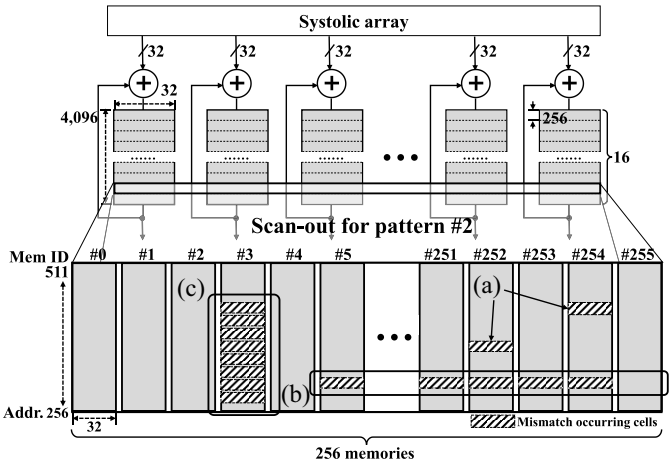

Fig. 7. Faulty types according to mismatch locations.

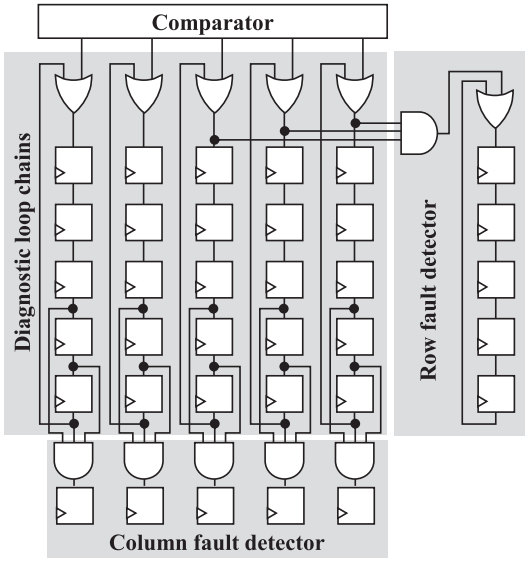

Fig. 8. Examples of self-diagnosis logic structure.

mode is set in hybrid BIST, and memories are tested using the fault-free hybrid BIST. It is because memories inside the accumulator block are used in scan tests on the systolic array. In this case, if unrepairable memory faults are detected, the chip also should be discarded. However, if repairable memory faults only exist, they are repaired before starting scan tests on the systolic array since the accumulator's memories are used in the scan tests. Finally, the LBIST mode is set in hybrid BIST, and scan tests on the systolic array are progressed. In this case, as the constructed scan chains should be fault-free for reliable scan tests on the systolic array, tests for them are preferentially performed. To verify fault-free scan chains, test patterns are just shifted into scan chains and out from scan chains without capture procedures. If there is a fault in scan chains, the fault information should exist in the output data of scan chains. After verifying fault-free scan chains in the systolic array, scan tests on each PE are progressed. After then, recovery is progressed if it facilitates to recover the systolic array.

C. Self-Diagnosis

Since faulty behavior varies depending on the fault location within a PE, specific fault types should be classified based on an analysis of faulty responses. If a MAC unit is faulty, the fault affects only the scan-out data of the corresponding PE, as shown in a) of Fig. 7. However, when a fault occurs in an activation register, its effect propagates from the defective PE to the right side, resulting in row-type failure, as shown in b) of Fig. 7. In this article, the fault in the activation register is called a row fault. On the other hand, if the weight or partial sum registers contain a fault, the fault effect propagates downward from the defective PE, resulting in column-type failure, as shown in c) of Fig. 7. In this article, the fault in the weight or partial sum registers is called a column fault.

In STRAIT, it is straightforward to locate faulty PEs. The scan-out data of each PE is stored in the memories assigned to the PE. Consequently, when the scan-out data is compared, the faulty PE can be identified simply by the memory ID and address at which the mismatch occurs. Fig. 8 shows self-diagnosis logic for a 5 × 5 systolic array. As shown, DLCs, which are composed of registers, are used to track the existence of errors for each PE in a systolic array. Whenever the scan-out data from a row of systolic array is compared with the expected data and the results are inserted in DLCs, data is circulated in each DLC. Until the test sequences are over, the data insertion in DLCs is continuously progressed and data circulation in DLCs is repeated. Suppose that error signal "1" is inserted from the comparator into one of the DLCs. In this case, the error signal is accumulated in the corresponding register by the OR gate. With the data circulation during the test, the diagnostic circuit is simultaneously driven and the locations of the faulty PEs are simply identical. In addition, to determine the locations of column and row faults, column and row fault detectors are implemented. If consecutive registers in a DLC are set to 1, it can be assumed that a column fault occurs. It can be simply checked by ANDing a few signals of the lowest registers of each DLC. Similarly, row fault occurrence can be checked by ANDing a few signals of registers, which are located on the right end side in DLCs. After the identification of fault locations and types, the fault information is stored in eNVM and used for fault recovery.

IV. Built-in Self-Recovery

A. PE Bypass Structure

The proposed recovery method of STRAIT relies on pruning faulty PEs. Pruning is a well-known technique that an efficient system can be developed by reducing the model size and the operation speed increases through the omission of unnecessary calculations [25], [26], [27]. In addition, pruning is more efficient for sparse networks than dense networks because it targets zero or near-zero weights, which have no effect on the calculation results. Indeed, many trained models exhibit a high level of sparsity [27], [28], [29], which is consistent with the properties of biological neural networks. Furthermore, studies for constructing a model with a high degree of sparsity have been also conducted [30], [31], [32], [33]. Therefore,

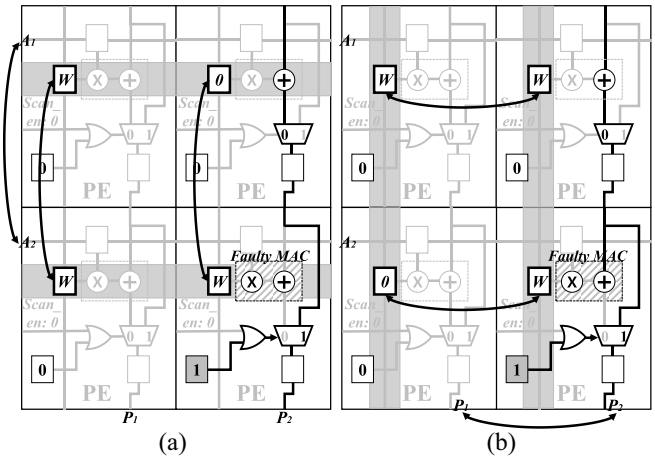

Fig. 9. Examples of weight swapping. (a) Row weight swapping and (b) column weight swapping.

sparse model pruning allows efficient system construction and operation, and it can be used to recover from faults.

In STRAIT, an additional multiplexer is already inserted in each PE for test. For this reason, it can be reused for pruning in fault recovery. Also, an OR gate with a disable register is inserted in each PE, as shown in Fig. 3. Therefore, the value stored in the disable register is decided by *PE_disable* signal, and the multiplexer can be controlled by the value since the value is inserted into multiplexer control part through an OR gate. For example, if faulty MAC units exist, the corresponding disable registers are set to 1 by *PE_disable* signal, and the MAC units are bypassed. In this case, as faulty PE information is already stored in eNVM after self-test and self-diagnosis, the values for pruning are loaded from eNVM by *PE_disable* signal before driving the systolic array.

B. Weight Swapping

If there are faults in MAC units, the corresponding disable registers are set to 1 by *PE_disable* signal, and the MAC units are bypassed. In this case, if the weight values for faulty PEs are zero, the values captured in the partial sum registers are equal to the values captured when there is no malfunction. However, if not, the values captured in the partial sum registers are changed, and causes the increase in error occurrence probability with the accumulation of different values. For this reason, weight swapping is progressed in STRAIT, as shown in Fig. 9. If a faulty PE and a PE with zero weight exist in the same column, their weights can be swapped as shown in Fig. 9(a). In this case, the weights of the entire row to which the faulty PE belongs should be swapped with the weights of the entire row that contains the zero-weight PE. Simultaneously, the activation values applied to both rows should be also swapped. Although the order of calculation is altered as the corresponding weights and activations are swapped, the final partial sum values stored in the accumulator are not changed. Therefore, it is possible to implement row weight swapping. Likewise, if a faulty PE and a PE with zero weight exist in the same row, their weights can be swapped as shown in Fig. 9(b). In this case, the weights of the entire column to which the faulty PE belongs should be swapped with the weights of the entire column that contains the zero-weight PE. However, the activation values do not need to be swapped in this case. Instead, as the locations of partial sum results are changed, it should be considered when the final calculation results are stored in an external memory. In addition, the sizes of weight and activation array are larger than the size of systolic array, the weight and activation arrays are separated into multiple tiles. In this case, the changed locations of partial sum results should be also considered when the partial sum results are reused as activations. Therefore, although column weight swapping can be used, it is recommended to implement row weight swapping.

Algorithm 1 represents the fault recovery flow using weight swapping. In weight swapping, only weight rows, which are expected to be inserted into faulty free rows of the systolic

Algorithm 1 Fault Recovery Algorithm Using Weight Swapping

recov_flag: recover flag for each faulty row in systolic array
num_row: the number of rows in systolic array
f_row_add: faulty row addresses in systolic array
num_cov_PE: the number of covered faulty PEs
num_f_row: the number of faulty rows in systolic array
faulty_position: faulty PE locations io each faulty row
z_weigth_position: zero weight locations in weight
f_count: the number of faulty PEs in each faulty row
recov_target: faulty row recover target in systolic array
mapping_result: faulty row recover result with the selected weight row

```
recov_flag = 0
for m = 0 to num_row − 1 do
    if m != f_row_add do
        // a weight row inserted into faulty free rows of the systolic array
        num_cov_PE = 0
        for n = 0 to num_f_row − 1 do
            if recov_flag[n] == 0 do // if a faulty row is not recovered yet
                if (faulty_position[n] match z_weight_position[m]) do
                    // if a faulty row can be recovered using the selected weight row
                    if num_cov_PE < f_count do
                        // if the number of recovered faulty PEs is larger
                        recov_target = n, num_cov_PE = f_count]
    if num_cov_PE != 0 do    // weight swapping
        mapping_result = (recov_target, m)
        mapping_result = (m, recov_target)
        recov_flag[recov_target] = 1
    else do
        mapping_result = (m, m)
if (all recov_flag = 1) do
    "Weigh swapping is successfully completed."
else do
    "Weight swapping is failed."
```

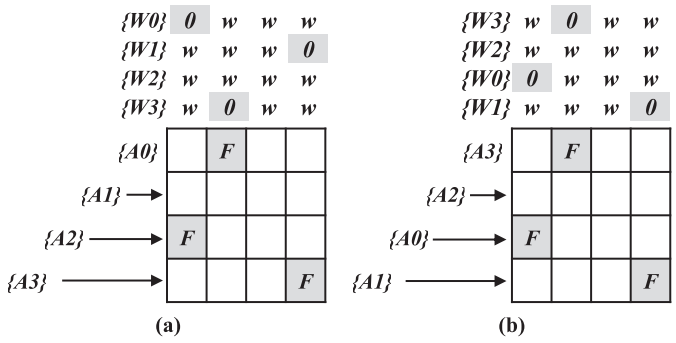

Fig. 10. Examples of weight allocation. (a) Before faulty recovery and (b) after faulty recovery.

array, are used to recover faulty rows of the systolic array. For this reason, when a weight row is selected at line 2, it is checked whether the corresponding row in the systolic array has a fault at line 3. If not, the weight row is considered for faulty row recovery in the systolic array. In this case, zero weight locations in the weight row are compared to faulty PE locations in each row of the systolic array at line 7. After then, if zero weight values of the weight row can be assigned to all faulty PE locations in a faulty row of the systolic array, the faulty row is selected as a recovery target. However, as a weight row can recover multiple faulty rows in the systolic array, the final recovery target should be selected among them. For this reason, the faulty row with the most faults is selected as the final recovery target among recoverable multiple faulty rows in the systolic array at lines 8 and 9. It is because the larger the number of faults in a faulty row is, the more difficult it is to recover the faulty row using other weight rows. Finally, if the final recovery target is selected after considering all faulty rows in the systolic array, weight swapping is progressed from lines 10 to 13. However, if the selected weight row cannot recover any faulty row in the systolic array, it is just assigned to the original row in the systolic array at line 15. If all faulty rows in the systolic array are recovered after all weight rows are considered, weight swapping is successfully completed at line 17. If not, the systolic array should be discarded or other recovery methods should be used at line 19.

C. Allocation

Although weight swapping can improve pruning efficiency for fault recovery, fault recovery is impossible if the number of faulty rows exceeds that of nonfaulty rows in the systolic array. To address the issue, weight allocation can be utilized in STRAIT. Fig. 10 shows the examples of weight allocation. "*F*" indicates the location of faulty PE, and *{Wn}* denotes the weights of each row. *{An}* denotes the activations to be inserted in each row. Weights were again detailed with "*0*" and nonzero values "*w*." In the example, the systolic array has faulty PEs in the first, third, and fourth rows, as shown in Fig. 10(a). In this case, as the number of nonfaulty rows is 1, weight swapping cannot be applied for fault recovery. However, as long as zero-weights are assigned to faulty PEs, fault recovery is possible. As shown in Fig. 10(b), although three rows in the systolic array are faulty, the systolic array can be used since zero-weights are allocated on faulty PE locations.

Algorithm 2 Fault Recovery Algorithm Using Weight Allocation

allo_flag: allocation flag for each row in systolic array
recov_flag: recover flag for each faulty row in systolic array
num_row: the number of rows in systolic array
num_cov_PE: the number of covered faulty PEs
num_f_row: the number of faulty rows in systolic array
faulty_position: faulty PE locations io each faulty row
z_weigth_position: zero weight locations in weight
f_count: the number of faulty PEs in each faulty row
recov_target: faulty row recover target in systolic array
recov_target_add: address of faulty row recover target in systolic array
mapping_result: faulty row recover result with the selected weight row

```
1: allo_flag = 0, recov_flag = 0
2: for m = 0 to num_row − 1 do
3:    num_cov_PE = 0
4:    for n = 0 to num_f_row − 1 do
5:       if recov_flag[n] == 0 do
                                   // if faulty row has not been recovered yet
6:          if (faulty_position[n] match z_weight_position[m]) do
                  // if faulty row can be recovered using the selected weight row
7:             if num_cov_PE < f_count do
                                   // if the number of recovered faulty PEs is larger
8:                recov_target = n, num_cov_PE = f_count
9:    if num_cov_PE != 0 do        // weight allocation
10:      mapping_result = (recov_target, m)
11:                  allo_flag[recov_target_add] = 1, recov_flag[recov_target] = 1
// if the selected weight row cannot be used for fault recovery
12: if num_cov_PE == 0 do
13:    for k = 0 to num_row - 1 do
14:       if allo_flag[k] == 0 do
15:          mapping_result = (k, m), allo_flag[k] = 1
16:          break
                        // the selected weight row is allocated on a fault free row
17: if (all allo_flag = 1) do
18:    "Weigh allocation is successfully completed."
19: else do
20:    "Weight allocation is failed."
```

Algorithm 2 represents the fault recovery flow using weight allocation. As shown, the fault recovery flow using weight allocation has similar processes to that using weight swapping. In weight allocation, all weight rows are used to recover faulty rows of the systolic array. For this reason, as soon as a weight row is selected at line 2, zero weight locations in the weight row are compared to faulty PE locations in each row of the systolic array at line 6. After then, if zero weight values of the weight row can be assigned to all faulty PE locations

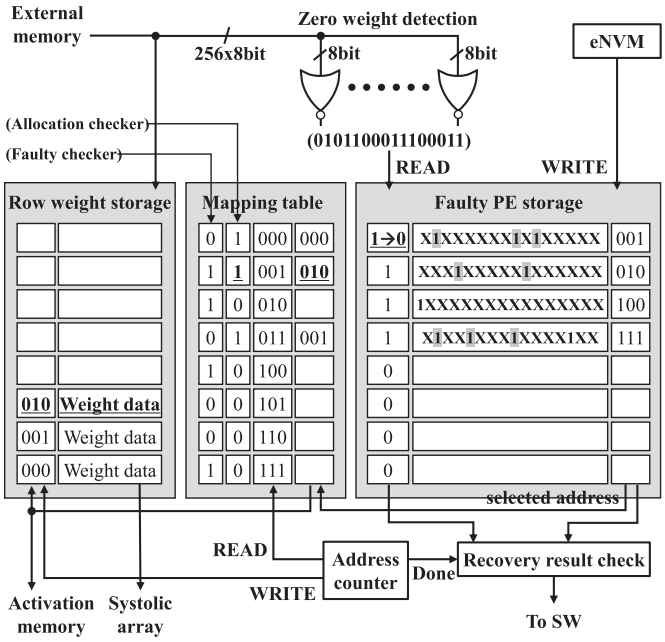

Fig. 11. Hardware structure for weight allocation.

in a faulty row of the systolic array, the faulty row is selected as a recovery target similar to weight swapping. Likewise, as a weight row can recover multiple faulty rows in the systolic array, the faulty row with the most faults is selected as the final recovery target among recoverable multiple faulty rows in the systolic array at lines 7 and 8. However, after all faulty rows in the systolic array are considered and the final recovery target is selected, only weight row assignment for the final recovery target is progressed at line 10 without weight swapping. It is different recovery process with weight swapping. In addition, if the selected weight row cannot recover any faulty row in the systolic array, it is just assigned to not the original row but a fault-free row in the systolic array at line 15. After then, if all faulty rows in the systolic array are recovered, weight allocation is successfully completed at line 18. If not, the systolic array should be discarded or other recovery methods should be used at line 20.

To implement weight allocation, the content addressable memory (CAM)-based hardware structure is proposed for self-recovery. Fig. 11 shows the hardware structure for weight allocation. As shown, it is composed of row weight storage, which already exists in the systolic array, mapping table, and faulty PE storage. Before starting self-recovery, faulty PE information is transferred from eNVM and stored in faulty PE storage. In this case, faulty PE locations are stored with the corresponding row addresses. As faulty PE storage is constructed with ternary CAM (TCAM), which supports don't care value (X), the faulty PE locations are represented as 1, and the other PEs are represented as X. After then, self-recovery processes are started. Whenever a row of weight data is transferred from external memory, it is sequentially stored with the corresponding row address in row weight storage. In addition, each weight data composed of multiple bits is simultaneously converted to 1-bit value by NOR gates to find zero-weight locations in the row of weight data. For this reason, if weights are zero, the corresponding bits are converted to 1; if not, the corresponding bits are converted to 0. After then, the locations of zero-weights are compared to faulty PE locations in faulty PE storage. Sequentially, a row address of which all faulty PE locations are matched with zero-weight locations is selected. If there are multiple row addresses of which all faulty PE locations are matched with zero-weight locations, the row address of which the number of faulty PEs is the most is selected. In Fig. 11, faulty PE locations in the first and second lines are matched with zero-weight locations of inserted row of weight data. However, as the number of faulty PEs in the first line is larger than that in the second line, the row address "001" is selected and the corresponding valid bit is set to 0. The comparisons and selections can be rapidly progressed because faulty PE storage is constructed using CAM. CAM supports that input data is compared to all internal contents and the matched data is out in single cycle. After row address selection in faulty PE storage, the selected address is transmitted to the mapping table and stored. In the example, as the row of weight data with "010" address is inserted from external memory and the row address "001" is selected in faulty PE storage, "010" is stored to allocate the row of weight data with "001" address in the mapping table, which means the row of weight data with "010" is inserted instead of that with "001" when weight data is transferred to the systolic array. Also, the corresponding allocation check bit is simultaneously set to 1. On the other hand, if a row address is not selected in faulty PE storage as there is no case that all faulty PE locations are matched with zero-weight locations, the row address of weight data inserted from external memory is stored to allocate the row of weight data that both the faulty checker bit and the allocation checker bit are 0 in the mapping table, and the corresponding allocation checker bit is set to 1. In the example, when the row of weight data with "000" address is inserted from external memory, it is stored to allocate the row of weight data with "000" address because its faulty checker bit and its allocation checker bit are 0. Likewise, when the row of weight data with "001" is inserted from external memory, it is stored to allocate the row of weight data with "011" address because of the same reason. After faulty recovery processes are finished, all weight data is transferred to the systolic array. In this case, the original address is sequentially transferred from the address counter to the mapping table, and the assigned address is rapidly selected. It is possible because the mapping table is also constructed using CAM. After then, the selected address is transferred to activation memory and row weight storage, and the corresponding row of weight data is transferred to the systolic array. During weight data is transferred to the systolic array, the next faulty PE information is transferred from eNVM and stored in faulty PE storage simultaneously. In addition, the next self-recovery processes are progressed during systolic array driving. For this reason, the performance degradation is negligible since they are pipelined.

D. Row Bypass With Column Skipping

Most of the literature on fault recovery have been concentrated on faulty MAC unit recovery [11], [15], [23], [24].

TABLE II
ATPG RESULT COMPARISONS

Fault type	SA fault		TD fault	
Architectures	[19]	STRAIT	[19]	STRAIT
Test coverage	100.00%	100.00%	86.42%	100.00%
Pattern counts	15	12	17	18

TABLE III
HARDWARE OVERHEAD COMPARISONS FOR SELF-TEST AND SELF-DIAGNOSIS

Architectures	[19]	STRAIT
BIST area	0.10%	0.01%
PE area	5.31%	5.24%
Diagnosis area	N.A.	2.50%
Total area	5.41%	7.75%

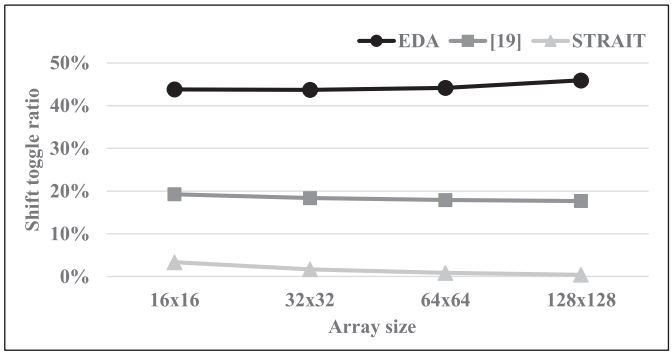

Fig. 12. Scan shift power comparisons.

However, when registers in PEs have faults, the effects of the faults propagate to the connected PEs, resulting in column or row faults. It can lead to more severe errors. In this case, disabling only the faulty PE does not resolve the issue. For this reason, a method for disabling all PEs in the row or column where the failure occurs is required. In the case of column faults, it can be a simple solution to disable the corresponding accumulator memories in which the operation results are stored and not to reuse them in the unified buffer. On the other hand, in the case of row faults, they are caused by an activation register problem. For this reason, all PEs in faulty row can be bypassed using *PE_disable* signal in STRAIT. For example, if row faults exist, disable registers in the corresponding rows where the failure occurs are set to 1 by *PE_disable* signal, and the entire rows are bypassed. However, it can cause performance degradation because a large number of PEs must be disabled. Therefore, it is preferable to recover faults using weight swapping and weight allocation while minimizing to disable accumulator memories for column skipping and bypass of the entire rows.

V. EXPERIMENTAL RESULTS

STRAIT can progress self-test, self-diagnosis, and self-recovery. However, since there is no method that can support all of them, each is evaluated, respectively. Experiments were conducted using Synopsys CAD tools and SAED 32 nm Library.

A. Self-Test and Self-Diagnosis

To evaluate the self-test and self-diagnosis methods in STRAIT, [19] was used since it has been the most recently proposed and its entire performance is better than other conventional methods. Table II represents the ATPG results of STRAIT and [19]. As shown, the SA fault test coverages of STRAIT and [19] are 100%, which is the optimal test coverage. However, the TD fault test coverage of [19] is only 86.4%. It can be critical when the systolic array is used for applications requiring accuracy and reliability, such as medical diagnosis, self-driving vehicles, security, and financial products. In this case, undetected faults can lead to severe problems. On the other hand, since the TD fault test coverage of STRAIT is 100%, STRAIT can reserve a reliable systolic array. Nevertheless, the number of test patterns is small regardless of fault type.

Table III represents the hardware overheads of STRAIT and [19]. As shown, the LBIST module for self-test of the systolic array is additionally implemented in [19], but its hardware overhead is negligible, which is only 0.10%. Likewise, as MBIST is modified for self-test of the systolic array in STRAIT, additional hardware for hybrid BIST is implemented, but its hardware overhead is also negligible, which is only 0.01%. Since the output comparator is additionally implemented in [19] but the MBIST module is slightly modified to construct hybrid BIST in STRAIT, the hardware overhead of BIST in STRAIT is smaller than that in [19]. In the case of PEs, additional hardware for scan configuration was considered. For fair comparisons, additional hardware for self-recovery is not considered in this section. As shown, the PE hardware overhead in STRAIT is also smaller than that in [19]. In both STRAIT and [19], MAC units are not included in scan configuration, and it is the same that weight, activation, and partial sum registers are synthesized as scan flip flops. The only difference is connecting structures of the partial sum scan chains. However, both hardware overheads in STRAIT and [19] are reasonable considering the entire systolic array. In addition, although self-diagnosis cannot be progressed in [19], it can be progressed in STRAIT with only 2.5% hardware overhead compared to a 256×256 systolic array. Although the total hardware overhead for self-test and diagnosis in STRAIT is 7.75% compared to the entire systolic array, it is negligible if it is compared to the entire chip area. For example, the hardware overhead is less than 1% compared to the entire chip of TPU.

Fig. 12 shows shift power comparisons. In general, scan tests are critical in power. Power consumption can be easily adjusted in the case of scan capture by separating or gating the clocks. On the other hand, power control is difficult during the scan shift operation, as all clocks in the target circuit must be toggled simultaneously. Additionally, data transitions tend to increase in order to facilitate testing. As a result, it is critical to minimize data toggling during scan shift operations. In the scan test, the toggle ratio of flip flops is frequently used to express power-related issues. As a result, scan shift power was compared using the average shift toggle ratio. As shown, the average shift toggle ratio of EDA basic scan structure [34] without special low-power options is approximately 45%. On

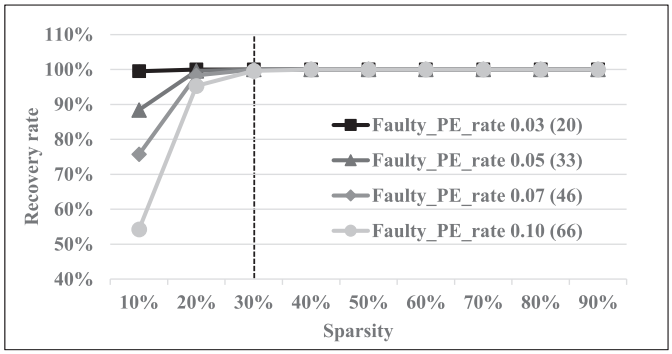

Fig. 13. Recovery rate according to sparsity.

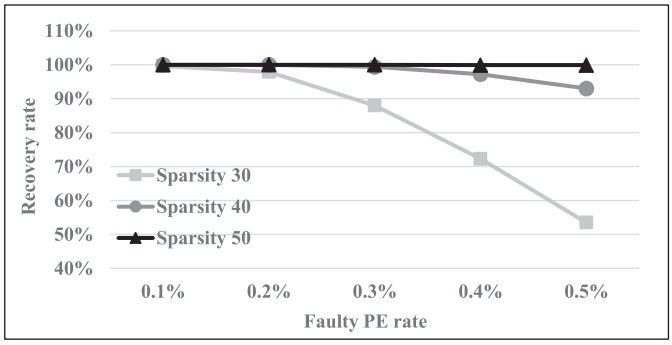

Fig. 14. Recovery rate according to faulty PE rate.

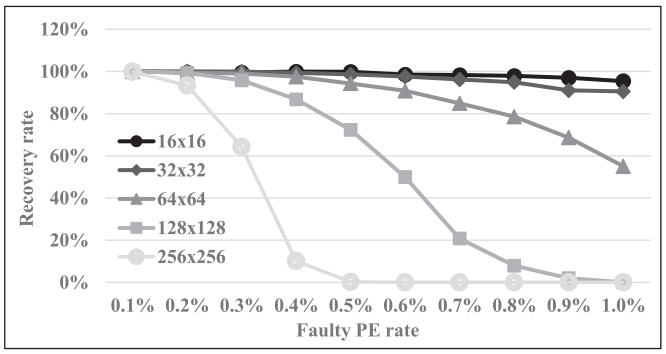

Fig. 15. Recovery rate according to array size.

the other hand, since weight and activation registers are constructed as scan chains, the average shift toggle ratio in [19] is reduced to less than 20% by reducing the shift toggle ratio in weight and activation scan chains. However, in STRAIT, since partial sum registers are also constructed as scan chains, the average shift toggle ratio is reduced to less than 5% by reducing the shift toggle ratio in weight, activation, and partial sum scan chains. Therefore, as the average shift toggle ratio is highly related to power consumption during the scan test, STRAIT can drastically reduce the power consumption during the scan test and it allows highly stable power in scan tests.

B. Self-Recovery

Experiments for self-recovery were conducted by increasing the faulty PE rate from 0.01% to 1.0% and changing the sparsity of weights from 10% to 90%. In addition, as the failure mechanism and recovery method vary depending on the fault location, faults were inserted by considering the area ratio of each part in PE. Since the area ratios of MAC units, partial sum registers, and weight/activation registers are 81%, 11%, and 4%, respectively, when PE is designed, faults were also inserted with the same ratio in each part. Also, each experiment was progressed 1000 times and the average values were calculated.

Fig. 13 shows the recovery rate according to the sparsity. In the experiments, faults were injected into a 256 × 256 systolic array with various faulty PE rates. As shown in Fig. 13, if the sparsity is larger than 30%, STRAIT can achieve a 100% recovery rate regardless of the faulty PE rate. However, if the sparsity is reduced lower than 30%, the recovery rate of STRAIT is reduced. It is because the number of rows for which all zero values are mapped to the required positions for fault recovery is reduced with the decrease of sparsity. However, since the trained weights are typically very sparse and various training methods increase sparsity without sacrificing accuracy [27], [28], sufficiently sparse networks can be secured.

Fig. 14 shows the recovery rate according to the faulty PE rate. In the experiments, faults were injected into a 256 × 256 systolic array with various sparsity. If the faulty PE rate is lower than or equal to 0.1%, STRAIT can achieve a 100% recovery rate regardless of the sparsity. On the other hand, if the faulty PE rate increases, the recovery rate is reduced. However, it is a rare case that the faulty PE rate becomes larger than 0.1%. It is because a PE is composed of many transistors. In other words, even when a lot of transistors have faults, a small number of faulty PEs just occur since most of them can be included in the same PEs. Therefore, faulty PE rate 0.1% already includes the cases that many faults occur in the systolic array.

Fig. 15 shows the recovery rate considering the array size. As shown, the recovery rate decreases with the increase in the array size. It is because considering weight allocation, the number of faulty PEs in one row increases with the increase of array size. Consequently, the recovery conditions become more complex, and in some cases, recovery is impossible. However, in this case, the recovery rate can increase by selecting a method for recovering row bypass instead of weight allocation.

Based on the experimental results of Figs. 13–15, if the expected fault rate and size of the systolic array to be used are reflected and a trained network with a sparsity that can reach the target recovery rate through sparsity-aware training is created and applied, it is possible to secure reliability by completely excluding inference errors caused by faults.

Fig. 16 shows PE utilization in STRAIT and [24]. In [24], even if a PE column has only one PE fault, all PEs in the corresponding column are unused. As a result, the PE utility can be drastically reduced. For this reason, a method that allows for faults in the lower bits (LBF), which does not significantly affect the inference error, has been also proposed in [24]. Therefore, in comparisons of PE utilization, the LBF ratio was changed from 10% to 30%, and PE utilization was evaluated according to the fault rate in the case of [24]. In addition, it was explained in [24] that the rows can be disabled,

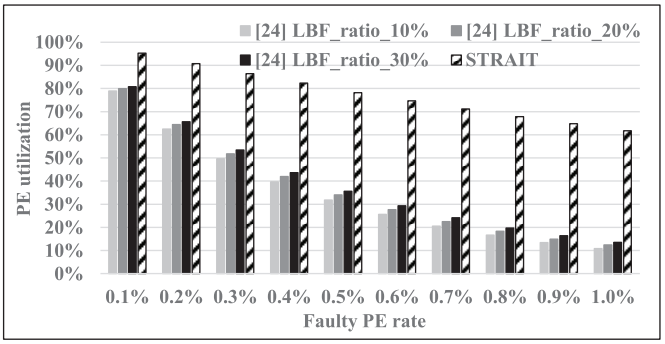

Fig. 16. Comparisons of PE utilization.

TABLE IV
HARDWARE OVERHEAD COMPARISONS FOR SELF-RECOVERY

Architectures	[11]	[24]	STRAIT
PE area	11.54%	0.00%	-2.51%
BISR area	0.00%	0.00%	0.1 %
Total area	11.54%	0.00%	-2.41%

but a function for completely bypassing the rows was missing. Therefore, only column disable method was used in the experiments. In STRAIT, fault types are checked, and weight allocation without performance degradation is used in the case of MAC unit faults. PE utilization is reduced only when column and row faults occur in STRAIT. Therefore, as shown in Fig. 16, the PE utilization of STRAIT is higher than that of the method proposed in [24], and the difference between them increases with the increase of the faulty PE rate.

The area overhead associated with the addition of the recovery function was compared to that in previous studies based on the premise that the scan test was preceded. For this reason, it is assumed in area comparisons that the systolic array has full scan structure for systolic array test. In [11], the MAC bypass logic and the full scan structure were considered. In [24], the area was measured by assuming only the full scan structure without any additional logic because fault recovery was handled at the software level. In STRAIT, the hardware overhead was reduced by sharing the mux logic for bypassing the MAC unit with the scan mux. Moreover, the scan overhead was reduced using the functional structure of weights and activation registers. Consequently, as summarized in Table IV, even though the logic for recovery was added, the chip area decreases by 2.5% compared to that of the full scan structure of [24]. In the case of weight allocation hardware, it was considered that since the SRAM cell size is 6T, CAM and TCAM generally have cell sizes of 10T and 16T, respectively. For this reason, assuming that the sizes of CAM and TCAM are twice and thrice compared to SRAM, respectively, the overhead by weight allocation hardware is estimated as only 0.1%. Therefore, self-recovery can be implemented with negligible hardware overhead in STRAIT.

VI. CONCLUSION

In this article, STRAIT is proposed to apply for a systolic array-based AI accelerator. It is an integrated solution that enables self-management from test to diagnosis and fault recovery. With STRAIT, it is possible to perform a stable test within a short time with a small hardware overhead. Moreover, STRAIT allows accurate fault localization and fault type analysis. By using the test and diagnosis results, accurate fault recovery is possible with less performance degradation. Therefore, STRAIT can contribute to securing the reliability of mission-critical applications by supporting self-test, self-diagnosis, and self-recovery features.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.
[2] J. Park et al., "Deep learning inference in Facebook data centers: Characterization performance optimizations and hardware implications," 2018, *arXiv:1811.09886*.
[3] M. Naumov et al., "Deep learning recommendation model for personalization and recommendation systems," 2019, *arXiv:1906.00091*.
[4] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ML models: A survey and insights," 2020, *arXiv:2007.00864*.
[5] S. Kundu, S. Banerjee, A. Raha, S. Natarajan, and K. Basu, "Toward functional safety of systolic array-based deep learning hardware accelerators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 48–61, Jan. 2020.
[6] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020.
[7] Y. Chen et al., "DaDianNao: A machine-learning supercomputer," in *Proc. IEEE/ACM Int. Symp. Microarchitect.*, Dec. 2014, pp. 609–622.
[8] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Architect. News*, vol. 44, no. 3, pp. 243–254, 2016.
[9] Y. Chen, T. Krishna, J. Emer, and V. Sze, "EyeRiss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, pp. 127–138, Jan. 2017.
[10] N. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. Int. Symp. Comput. Architect.*, 2017, pp. 1–12.
[11] J. Zhang, T. Gu, K. Basu, and S. Garg, "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Des. Test*, vol. 36, no. 5, pp. 44–53, Oct. 2019.
[12] A. Gebregiorgi and M. Tahoori, "Testing of neuromorphic circuits: Structural vs. functional," in *Proc. IEEE Int. Test Conf.*, 2019, pp. 1–10.
[13] A. Chaudhuri, J. Talukdar, F. Su, and K. Chakrabarty, "Functional criticality classification of structural faults in AI accelerators," in *Proc. IEEE Int. Test Conf.*, 2020, pp. 1–5.
[14] S. Kundu, A. Soyyiğit, K. Hoque, and K. Basu, "High-level modeling of manufacturing faults in deep neural network accelerators," in *Proc. IEEE Int. Symp. On-Line Test. Robust Syst. Design*, 2020, pp. 1–4.
[15] S. Kundu et al., "Special session: Reliability analysis for ML/AI hardware," 2021, *arXiv:2103.12166*.
[16] H. Ma et al., "A case study of testing strategy for AI SoC," in *Proc. IEEE Int. Test Conf. Asia*, 2019, pp. 61–66.
[17] A. Chaudhuri, C. Liu, X. Fan, and K. Chakrabarty, "C-testing of AI accelerators," in *Proc. IEEE 29th Asian Test Symp.*, 2020, pp. 1–6.
[18] A. Chaudhuri, C. Liu, X. Fan, and K. Chakrabarty, "C-testing and efficient fault localization for AI accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 7, pp. 2348–2361, Jul. 2022.
[19] U. Solangi, M. Ibtesam, M. Ansari, J. Kim, and S. Park, "Test architecture for systolic array of edge-based AI accelerator," *IEEE Access*, vol. 9, pp. 96700–96710, 2021.
[20] T. Koal, H. Vierhaus, and D. Scheit, "A concept for logic self-repair," in *Proc. IEEE Euromicro Conf. Digit. Syst. Design Archit. Methods Tools*, 2009, pp. 1–6.
[21] J. Kim and S. M. Reddy, "On the design of fault-tolerant two-dimensional systolic arrays for yield enhancement," *IEEE Trans. Comput.*, vol. C-38, no. 4, pp. 515–525, Apr. 1989.
[22] K. Cho, I. Lee, H. Lim, and S. Kang, "Efficient systolic-array redundancy architecture for offline/online repair," *Electronics*, vol. 9, no. 2, p. 338, 2020.
[23] J. Zhang, T. Gu, K. Basu, and S. Garg, "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *Proc. IEEE VLSI Test Symp.*, 2018, pp. 1–8.

[24] M. Sadi and U. Guin, "Test and yield loss reduction of AI and deep learning accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 1, pp. 104–115, Jan. 2022.
[25] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.
[26] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," 2019, *arXiv:1902.09574*.
[27] D. Mocanu, E. Mocanu, P. Stone, and P. Nguyen, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nat. Commun.*, vol. 9, pp. 1–12, Jun. 2018.
[28] S. Srinivas, A. Subramanya, and R. Babu, "Training sparse neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2017, pp. 455–462.
[29] J. Liu, Z. Xu, R. Shi, R. Cheung, and H. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," 2020, *arXiv:2005.06870*.
[30] S. Alford, R. Robinett, L. Milechin, and J. Kepner, "Pruned and structurally sparse neural networks," in *Proc. IEEE MIT Undergrad. Res. Technol. Conf.*, 2018, pp. 1–7.
[31] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.
[32] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," 2021, *arXiv:2102.00554*.
[33] C. Louizos, M. Welling, and D. Kingma, "Learning sparse neural networks through L0 regularization," 2017, *arXiv:1712.01312*.
[34] "Synopsys TestMAX DFT." Accessed: Sep. 7, 2021. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-dft.html

Hayoung Lee (Graduate Student Member, IEEE) received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2016, where he is currently pursuing the combined Ph.D. degree with the Department of Electrical and Electronic Engineering.

His current research interests include high-speed memory test, built-in self-testing, built-in self-repair, redundancy analysis algorithms, reliability, error detection/correction, system-level test and validation, design for testability/debug, and VLSI design.

Jihye Kim received the B.S., M.S., and Ph.D. degrees in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2002, 2004, and 2022, respectively.

Since 2004, she has been a DFT Engineer with Samsung Electronics, Hwaseong, South Korea, where she is currently a Principal Engineer. Her current research interests include VLSI/SOC design and testing, design for testability, silicon failure analysis, fault tolerance, and AI accelerator.

Jongho Park received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2020, where he is currently pursuing the combined Ph.D. degree with the Department of Electrical and Electronic Engineering.

His current research interests include design for testability, low-power testing, and built-in self-testing.

Sungho Kang (Senior Member, IEEE) received the B.S. degree in control and instrumentation engineering from Seoul National University, Seoul, South Korea, in 1986, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 1988 and 1992, respectively.

Since 1994, he has been a Professor with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul. He was a Research Scientist with the Schlumberger Laboratory for Computer Science, Schlumberger Inc., Austin, and a Senior Staff Engineer with Semiconductor Systems Design Technology, Motorola Inc., Austin. His current research interests include VLSI/SoC design and testing, design for testability, design for manufacturability, and fault-tolerant computing.