PA-2SBF: Pattern-Adaptive Two-Stage Bloom Filter for Run-time Memory Diagnostic Data Compression in Automotive SoCs

Sunyoung Park[1,2], Hyunji Kim[1,2], Hana Kim[1,2] and Ji-Hoon Kim[1,2]
[1]Department of Electronic and Electrical Engineering, Ewha Womans University, Seoul, Republic of Korea
[2]Graduate Program in Smart Factory, Ewha Womans University, Seoul, Republic of Korea
psuny0412@gmail.com

***Abstract*—In the realm of safety-critical Automotive System-on-Chip (SoC) design, memory functionality plays a crucial role in determining overall die yield due to its sizable footprint on the chip. Therefore, efficient methodologies for both post-manufacturing offline testing and real-time monitoring are essential to provide timely diagnostic feedback. This paper proses a real-time memory diagnostic data compression technique Pattern-Adaptive Two-Stage Bloom Filter (PA-2SBF) for automotive System-on-Chip (SoC) applications. PA-2SBF is designed to address the challenge of false positives incorporating frequently encountered failure patterns. Bloom filters, a probabilistic data structure renowned for their space-efficiency and quick approximate membership queries, are employed to expedite fault memory diagnosis information lookup and compression. Furthermore, failure patterns are considered to mitigate the false positive rate inherent in Bloom filters. The paper also presents a strategy for leveraging the compressed diagnostic information during run-time. Specifically, it exploits the quick lookup feature of Bloom filter to prohibit CPU access to defective memory regions, enhancing overall system reliability.**

***Index Terms*—Memory Diagnosis, Automotive SoC, Memory Testing, Test Compression, Bloom Filter.**

I. Introduction

In recent years, as the proportion of memory within automotive System-on-Chips (SoCs) has increased, the impact of memory on yield has grown [1], [2]. Hence, it is crucial to accurately diagnose memory failures and analyze diagnostic information. A widely used technique for illustrating failure constellations after a memory test is the bit fail map representation [3]. In this approach, memory is represented as a matrix, where each element corresponds to a physical cell location. Each bit within this matrix is assigned a value of 0 to indicate normal functionality or a value of 1 to signify the detection of a fault. Prior studies have employed a technique that involves the compression of diagnostic data acquired via Memory Built-In Self-Test (MBIST). This compressed information is subsequently transmitted to an external system for further analysis to mitigate the identified issue, where there is no mechanism to restrict access to failed memory regions during normal operation even though after online testing. This poses a significant challenge for automotive SoCs, which have stringent requirements for high functional safety during operation. Consequently, there is an imperative need for a novel approach capable of real-time integration of fault information to facilitate immediate response mechanisms for memory access.

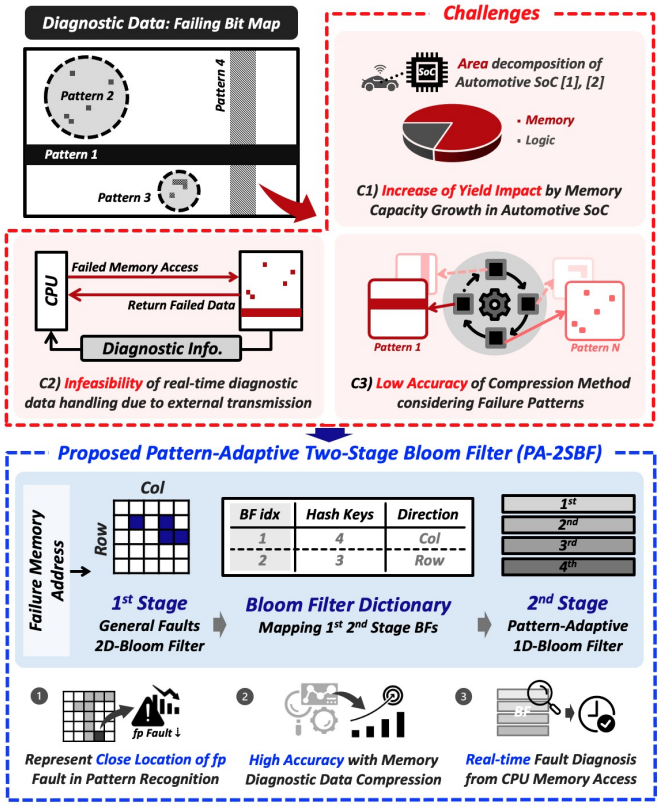

Fig. 1. Overview of proposed run-time memory diagnostic data compression Pattern-Adaptive Two-Stage Bloom Filter (PA-2SBF) process

In this paper, Bloom filters are employed for the compression of fault diagnosis information and rapid data retrieval. A Bloom filter is defined by a bit array of length m and k hash functions, serving as a membership filter. The Bloom filter sets k positions to 1 using these hash functions. To determine the presence of an element, the Bloom filter only needs to confirm if these k positions are set to 1. Consequently, Bloom filters enable the storage of significant data volumes within a constrained memory capacity. This inherent efficiency in memory utilization makes Bloom filters invaluable in a

979-8-3503-4859-0/DATE24/© 2024 EDAA

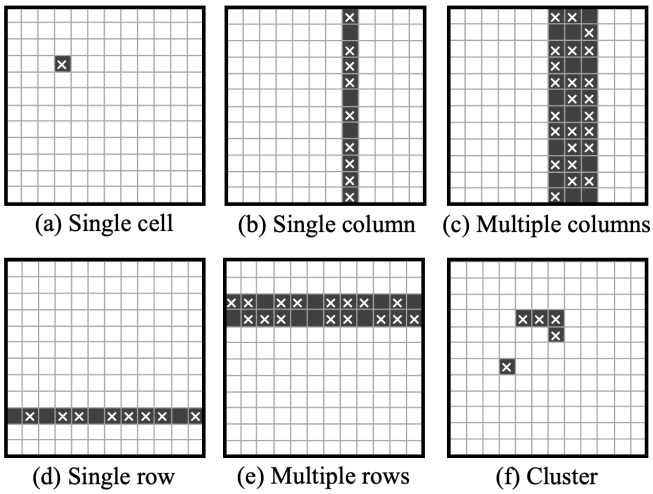

Fig. 2. Failure pattern examples; (a) Single cell, (b) Single column, (c) Multiple columns, (d) Single row, (e) Multiple rows, (f) cluster

multitude of domains, including computer networking [4], [5], security [6], big data [7], and cloud computing [8]. Due to the absence of false negatives, there is an advantage of not erroneously classifying faulty memory as functional memory. However, Bloom filters inherently exhibit a disadvantage in the form of a false positive rate, and an increase in memory capacity is necessary to overcome this issue. Therefore, a key consideration when using Bloom filters is how to minimize memory overhead while reducing the false positive rate.

This paper presents a run-time memory diagnostic data compression using Bloom filters for automotive SoC. As shown in Fig. 1, Pattern-Adaptive Two-Stage Bloom Filter (PA-2SBF) is proposed to efficiently compress memory diagnostic information and optimize the 2-stage Bloom filter through a Bloom filter dictionary. To mitigate the inherent false positive rate and reduce area overhead, the PA-2SBF method incorporates commonly used failure patterns, as illustrated in Fig. 2 [9], [10], [11], during the insertion of memory diagnostic information into the Bloom filter. These patterns encompass various scenarios, including single-cell failures, failures in a column/row, failures in multiple columns/rows, and cluster failures. Additionally, it dynamically maps the bit array based on patterns using an adaptive approach. Hence, even when false positives occur, there are advantages in marking faults close to the patterns during pattern recognition.

In contrast to previous approaches, this method offers several advantages. Firstly, it represents the precise locations, rather than approximate positions, primarily for pattern recognition. Additionally, it leverages the rapid Bloom filter lookup during run-time scenarios, eliminating the need to transmit data externally for analysis and response. This enables the CPU to perform real-time memory failure verification when accessing memory. While Bloom filters may exhibit false positives, they guarantee the absence of false negatives, making it advantageous for safety-critical applications like automotive systems to prevent safety-related incidents.

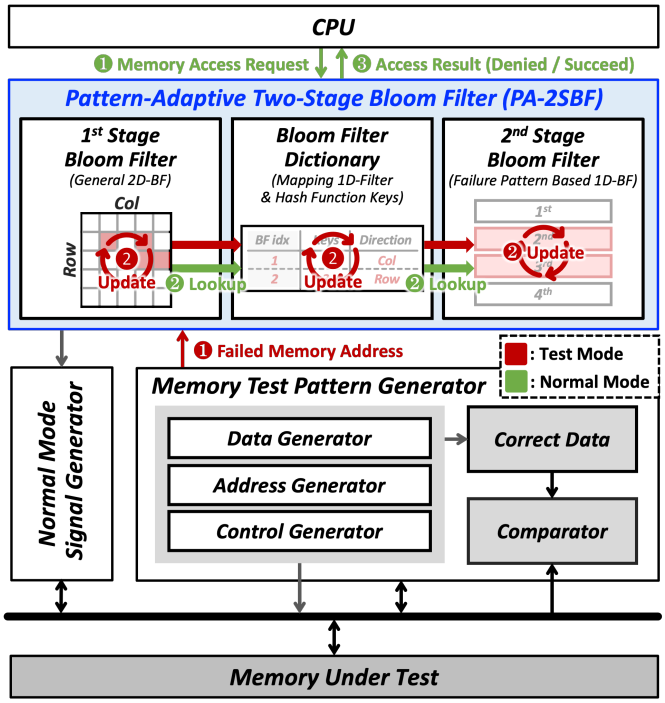

Fig. 3. Overview of proposed run-time memory diagnostic data compression architecture

As a result, unlike conventional approaches, this proposed methodology integrates information in real-time, effectively preventing access to failed memory cells.

II. Related Works

In the study conducted by Landzberg et al., [3] diagnostic information accumulation employed a straightforward approach, storing failure data in the form of a bit failing map. However, this method has a significant drawback as it consumes a substantial amount of memory space since each memory location is individually represented.

The march test algorithm is a widely adopted choice due to its advantageous combination of good fault coverage and low complexity. In the works of Li et al. [12] and Hou et al. [13], diagnostic data from the march test underwent compression using a syndrome scheme. Nevertheless, this approach comes with its own limitation: it necessitates content addressable memory for data accumulation and is restricted to the outcomes of the march algorithm. With the progression of deep sub-micron technology [14], the importance of more intricate memory testing algorithms has grown. For instance, as reported in [15], the galloping pattern was employed to detect failures that eluded detection in the march pattern.

John et al. [16] introduced a technique aimed at facilitating embedded memory diagnosis through the compression of test responses and automated bitmap recognition. However, this approach has a drawback in terms of the accuracy of the reconstructed constellation based on the compressed data.

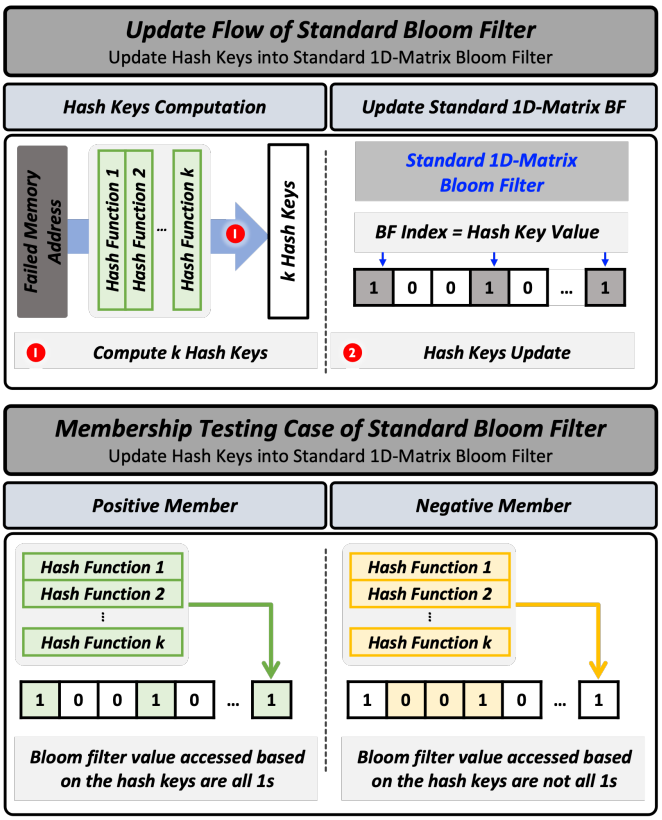

Fig. 4. Conventional Bloom filter update flow and membership testing cases

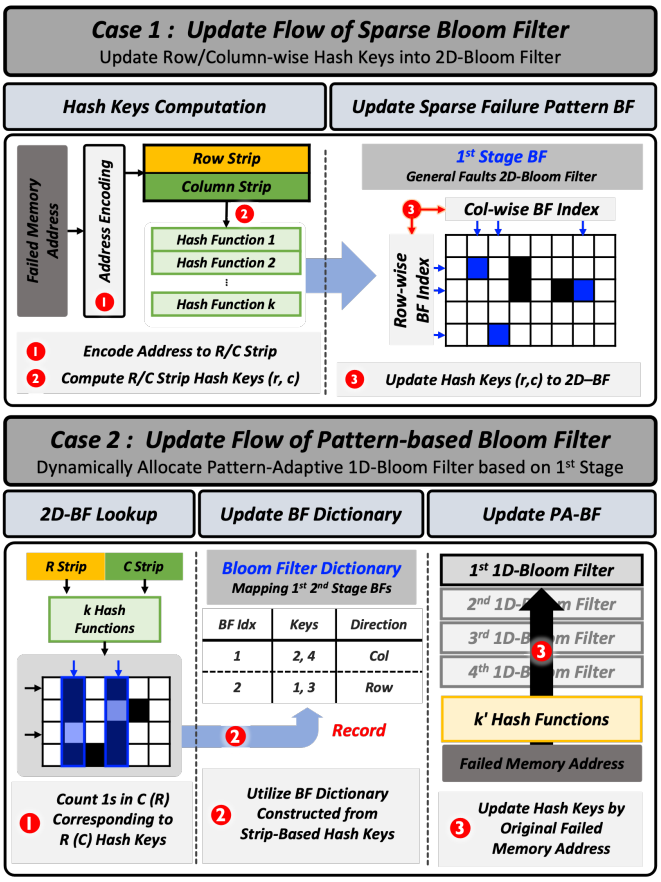

Fig. 5. Pattern-Adaptive Two-Stage Bloom Filter (PA-2SBF) failed memory address insertion process

Bernardi et al. [1] adopted a strategy where failure information was stored in a color-segment encoding format unable to accurately show the exact location of the fault occurrence, reducing memory usage, and subsequently conducting pattern recognition based on this format. Nonetheless, this method has the disadvantage of consuming excessive memory capacity when faults occur sparsely.

Insinga et al. [2] presented an algorithm that employs a coordinates-based compression approach, partitioning the memory logically into equally divided sectors referred to as pixels, each composed of a configurable number of word lines and bit lines. However, this method primarily focused on visualizing failure patterns for fault detection, which presents challenges in accurately representing the exact fault locations and consequently leads to reduced accuracy.

III. Pattern-Adaptive Two-Stage Bloom Filter

Fig. 3 presents the overview architecture assumed in this paper. Positioned between components such as the CPU and the memory test pattern generator, which includes MBIST, is the proposed PA-2SBF. It consists of the following components: a 1^{st} Stage Bloom Filter (2D-BF) that stores faults occurring in general situations, a 2^{nd} Stage Bloom Filter (1D-BF) allocated based on failure patterns, and a Bloom Filter Dictionary (BF Dictionary) responsible for mapping 1D-BF and hash function keys along with directions. Furthermore, PA-2SBF operates in two modes: the test mode, serving as a state for memory functionality verification, and the normal mode, representing the state when memory is accessed by the CPU for read or write operations. In test mode, PA-2SBF accepts failed memory addresses from the memory test pattern generator and inserts them as elements into PA-2SBF. In normal mode, when the CPU sends a memory access request, PA-2SBF performs memory testing to determine the accessibility of the requested address and provides the relevant information.

A. Conventional Bloom Filter

A Bloom filter is a data structure used for quick membership testing of data elements, commonly in databases, caches, and network routing. It excels in providing rapid data retrieval and efficient memory usage, making it highly advantageous for scenarios demanding quick data inquiries, especially when handling large datasets. This structure uses a fixed-size bit array initialized with all elements set to zero. During data insertion, multiple hash functions are applied to generate hash values, setting specific bits in the array to 1 as depicted in Fig. 4.

When verifying data membership, the same process of employing multiple hash functions is repeated, and the bit array indices are checked. If all bits at these indices are set to 1, the data is confirmed as a member of the dataset.

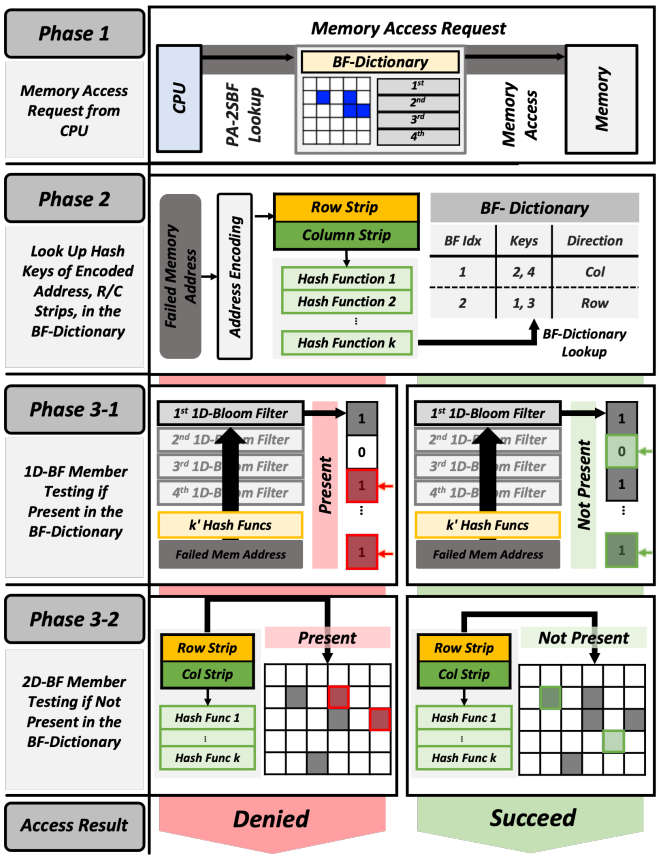

Fig. 6. Pattern-Adaptive Two-Stage Bloom Filter (PA-2SBF) failed memory address run-time lookup process

However, if any bit is 0, the data is considered not part of the set. In essence, a Bloom filter is a powerful tool for quick data retrieval and efficient memory use, particularly beneficial for sifting through extensive data volumes. It's important to note that Bloom filters may occasionally yield false positives, necessitating additional data validation during use.

While Bloom filters offer fast query capabilities, they are susceptible to false positives, where data not in the set may be incorrectly identified as a member. This susceptibility arises due to the use of multiple hash functions that can lead different data elements to hash to the same positions. When using a Bloom filter, considering the potential for false positives is essential. In this paper, we propose a 2-stage Bloom filter operation to address this issue.

B. Test Mode : PA-2SBF Element Insertion Flow

In test mode, when the memory test pattern generator detects a fault during memory testing and identifies a malfunction at a specific address, it forwards this address to PA-2SBF for compression. The address insertion flow can be observed in two cases, as shown in Fig. 5.

In case 1, the 1^{st} Stage BF is employed as a cache to store information without considering faulty patterns, which can occur in scenarios such as when single cells are scattered. To perform the hash keys computation process, the actual row/column addresses are first encoded into row/column strips. A strip represents an address as a range, rather than an exact address. For example, setting the address-to-strip ratio as 2:1 means that when the row address is 1 or 2, the row strip is represented as 1. After the encoding process, the row and column strips are each processed by k hash functions, yielding k pairs of (RowKey, ColKey) hash key coordinates. It checks the number of 1s corresponding to (RowKey_1, RowKey_2, ..., RowKey_k) and (ColKey_1, ColKey_2, ..., ColKey_k) on the 2D-BF to determine if there are previously stored faults in each of the corresponding row and column strips. In this scenario, if there were no previously stored faults, the 2D-BF bits corresponding to the k hash key coordinates (RowKey, ColKey) are set to 1s.

Case 2 involves the inference of pattern direction based on the 1^{st} stage 2D-BF and the subsequent update of the 2^{nd} stage 1D-BF and BF-Dictionary. Initially, to perform a lookup in the 2D-BF, *k'* pairs of strip hash key coordinates (RowKey,ColKey) are computed using the same methodology as in case 1. The examination focuses on the count of 1s associated with (RowKey_1,RowKey_2, ...,RowKey_k) and (ColKey_1,ColKey_2,...,ColKey_k) on the 2D-BF to ascertain whether previously recorded faults exist within the corresponding row and column strips. In the event of pre-existing faults within the corresponding row or column strips, the BF-Dictionary logs the hash keys linked to the direction and assigns a 1D-BF accordingly. Subsequently, faults occurring within the corresponding strips are updated in a pattern-adaptive manner, achieved by utilizing *k'* hash functions to derive memory address hash keys, thereby enhancing the overall accuracy.

C. Normal Mode : PA-2SBF Look-up Flow

In normal mode, which represents a typical situation where the memory is accessed for reading and writing, memory addresses are forwarded to PA-2SBF to check for faults at the desired memory address. The run-time flow for looking up PA-2SBF is composed of three phases, as shown in Fig. 6.

In phase 1, the CPU initiates memory access requests to perform read or write operations in memory. To access memory, these requests must undergo the PA-2SBF lookup process. Moving on to phase 2, it encompasses the conversion of memory addresses into strips and a subsequent verification process to determine whether the corresponding strip value has been assigned to a 1D-BF within the BF-Dictionary.

If the corresponding strip is found in the BF-Dictionary, phase 3-1 is initiated. It conducts a lookup on the corresponding 1D-BF within the BF-Dictionary using the key obtained from the memory address through *k'* hash functions. If the memory address is a member, indicating the presence of a fault at that specific address, memory access is denied to block further access. Conversely, when the lookup on the corresponding 1D-BF in the BF-Dictionary, using the hash keys, confirms that it is not a member, it signifies that the cell corresponding to the address is functioning normally. Memory access is deemed successful, and the request is forwarded to

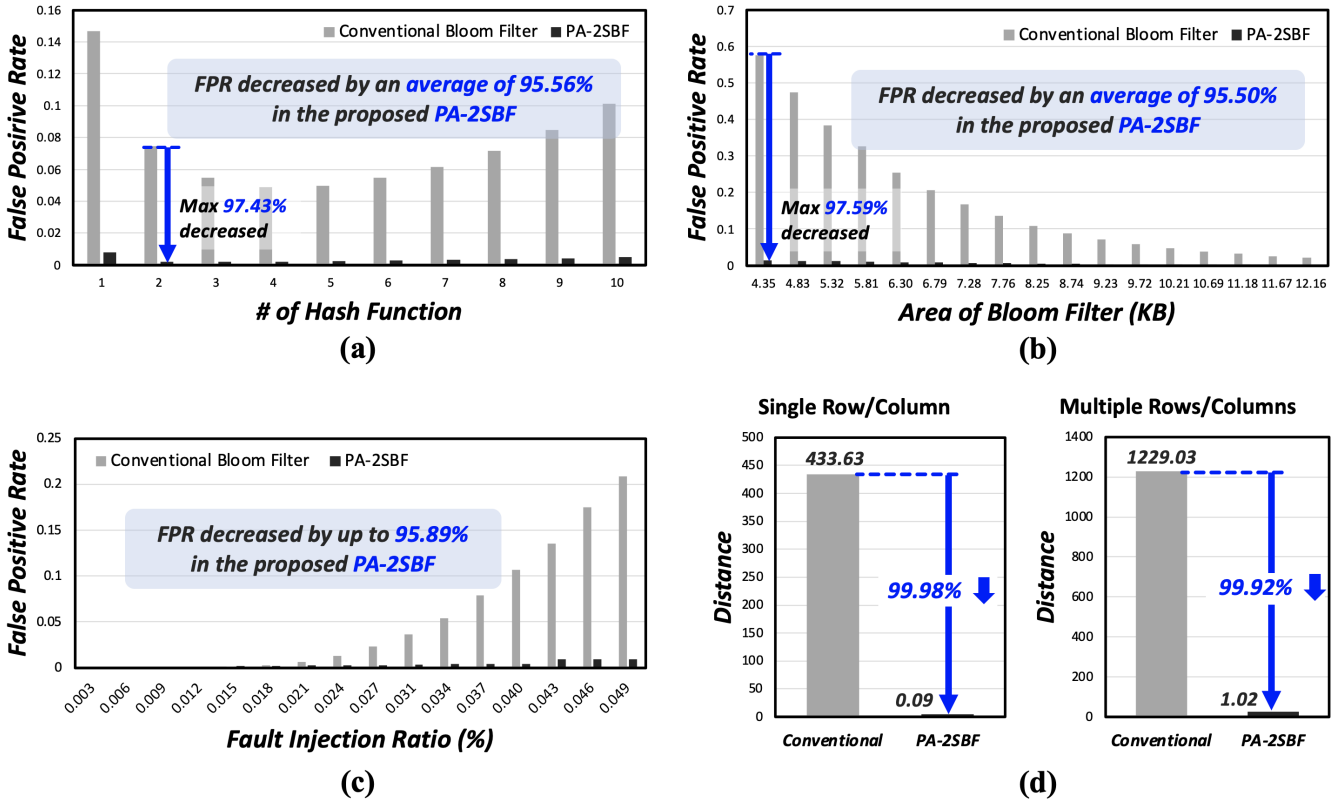

Fig. 7. Comparison between conventional Bloom filter and PA-2SBF by false positive rate; (a) false positive rate as per number of hash function, (b) false positive rate as per area of Bloom filter, (c) false positive rate as per fault injection ratio, (d) distance of single, multiple row/column

the memory. This process enhances the system's ability to accurately determine the status of memory addresses during run-time operations.

If the corresponding strip is not found in the BF-Dictionary, phase 3-2 is entered. During phase 3-2, a membership test is conducted on the 2D-BF using the computed k strip hash keys (RowKey, ColKey) from phase 2. If the test results indicate membership, it is inferred that a fault exists at the memory address, leading to the denial of memory access to prevent further operations. Conversely, by employing the k strip hash keys (RowKey, ColKey) computed during phase 2, a membership test is executed on the 2D-BF. In cases where the test identifies the strip as a non-member, it signifies the absence of faults at the corresponding memory address, thus enabling the smooth forwarding of memory access requests to the memory. This multi-phase process significantly enhances the system's ability to accurately assess memory address status during run-time operations.

IV. Experimental Results

In this section, we analyze the false positive rate based on Bloom filter parameters. The false positive rate (p) in a Bloom filter can be determined by the number of hash functions (k), the size of the bit array (m) and the number of elements (n). In this context, false positives are assumed to occur when faults are detected even when the memory is functioning correctly, potentially degrading the overall system performance. Therefore, minimizing these false positives is crucial to ensure the proper functioning of the entire system.

In order to evaluate the performance of our proposed PA-2SBF, we conducted a comprehensive analysis in a 4MB memory environment. In Fig. 7(a), an in-depth analysis of the false positive rate is conducted, taking into account the number of hash functions used. The proposed method demonstrates a substantial improvement over conventional Bloom filters, averaging from 95.6% to a maximum of 97.4% enhancement. Additionally, it is noted that the false positive rate does not vary proportionally with the value of k, whether it is small or large. Therefore, it is essential to configure an appropriate value based on the desired scenario.

In Fig. 7(b), the relationship between the allocated memory capacity in the Bloom filter and the false positive rate is depicted. PA-2SBF distinguishes itself by segmenting the bit array and allocating 1D-BFs based on patterns. The 2D-BF in the 1^{st} stage is fixed at 2.39KB, allowing adjustments to the size of the 1D-BF in experiments. When considering the occurrence of multiple row/column fault patterns, the false positive rate significantly improves, averaging from 95.5% to a maximum of 97.6%, compared to a conventional Bloom filter using the same total size of Bloom filter.

Fig. 7(c) depicts the variation in false positive rates as the number of faults stored in the conventional Bloom filter and

TABLE I
COMPARISON WITH PRIOR WORKS USING DIAGNOSTIC DATA COMPRESSION

	[3]	[1]	[2]	Baseline	This work
Compression Format	Bit-by-Bit	Colored-Slice	SLAC Pixel	Conventional Bloom Filter	PA-2SBF
Failure Pattern Consideration	No	Yes	Yes	No	Yes
Compression_{a} Efficiency	1	1.38	1.58	0.36	0.13
Fail Bitmap Reconstruction Rate	100%	Low	Low	94.7%	99.9%
Run-time Operation	No	No	No	Yes	Yes

[a]Worst case scenario is assumed and calculated the ratio based on [3]

PA-2SBF. When the number of occurring faults is low, the false positive rate of the conventional Bloom filter is lower. This is because the 1^{st} stage Bloom filter records the initial faults in approximate locations. However, as the number of faults increases, it becomes evident that the proposed method demonstrates a superior false positive rate. For example, in a scenario where faults occur in rows and columns, resulting in faults accounting for 0.04% of the total memory area, the proposed method demonstrates a significant improvement of a maximum 95.9% compared to the conventional Bloom filter.

Achieving a high accuracy in failed bitmap reconstruction is of paramount importance for diagnostic data compression and external analysis. To enhance the precision of pattern recognition commonly performed with diagnostic data, our proposed method goes beyond the mere reduction of the false positive rate. It is specifically designed to leverage the characteristics of a Bloom filter, ensuring that if false positives do occur, they manifest in physically proximate locations to actual faults rather than randomly. Illustrated in Fig. 7(d), especially in scenarios where a single fault occurs in either a row or column, false positive elements in PA-2SBF are predominantly associated with the same row and column. Moreover, in situations involving multiple row and column faults, PA-2SBF demonstrates a 1229 times improvement.

Table I provides a detailed comparative analysis, drawing distinctions between previous works and our proposed scheme, PA-2SBF. This innovative approach is tailored to support diverse memory configurations while specializing in addressing failure patterns. In the worst-case scenario, considering the compression area requirements, PA-2SBF consumes a mere 13% of the resources compared to the reference method denoted as [3]. Furthermore, it achieves a reconstruction rate of 99.9%. Importantly, it distinguishes itself from other approaches by enabling real-time adaptation during run-time scenarios.

V. CONCLUSION

Efficient methodologies for post-manufacturing offline testing and real-time monitoring are essential in providing timely diagnostic feedback for automotive SoC designs, where memory functionality significantly impacts overall die yield. This paper introduces a run-time memory diagnostic data compression technique using PA-2SBF, which leverages Bloom filters to reduce the data volume of diagnostic information. To address the inherent false positive issue in Bloom filters, this proposed method considers failure patterns. Additionally, the paper outlines a strategy for utilizing the compressed diagnostic information during run-time. Specifically, it takes advantage of the Bloom filter's quick lookup capability to prevent CPU access to defective memory regions, thereby enhancing the overall system reliability.

ACKNOWLEDGEMENTS

This paper was result of the research project supported by SK hynix Inc.

REFERENCES

[1] P. Bernardi, G. Insinga, G. Paganini, R. Cantoro, P. Beer, M. Coppetta, N. Mautone, G. Carnevale, P. Scaramuzza, and R. Ullmann, "Optimized diagnostic strategy for embedded memories of automotive systems-on-chip," in *2022 IEEE European Test Symposium (ETS)*, pp. 1–6, 2022.

[2] G. Insinga, M. Battilana, M. Coppetta, N. Mautone, G. Carnevale, M. Giltrelli, P. Scaramuzza, and R. Ullmann, "Density-oriented diagnostic data compression strategy for characterization of embedded memories in automotive systems-on-chip," in *2023 IEEE European Test Symposium (ETS)*, pp. 1–6, IEEE, 2023.

[3] A. Landzberg, *Microelectronics Manufacturing Diagnostics Handbook*. Springer Science & Business Media, 2012.

[4] J. H. Mun and H. Lim, "New approach for efficient ip address lookup using a bloom filter in trie-based algorithms," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1558–1565, 2015.

[5] H. Lee and A. Nakao, "Improving bloom filter forwarding architectures," *IEEE Communications Letters*, vol. 18, no. 10, pp. 1715–1718, 2014.

[6] R. Patgiri, S. Nayak, and S. K. Borgohain, "Preventing ddos using bloom filter: A survey," *arXiv preprint arXiv:1810.06689*, 2018.

[7] R. Patgiri, S. Nayak, and S. K. Borgohain, "Role of bloom filter in big data research: A survey," *arXiv preprint arXiv:1903.06565*, 2019.

[8] A. Singh, S. Garg, K. Kaur, S. Batra, N. Kumar, and K.-K. R. Choo, "Fuzzy-folded bloom filter-as-a-service for big data storage in the cloud," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2338–2348, 2018.

[9] J. Segal, A. Jee, D. Lepejian, and B. Chu, "Using electrical bitmap results from embedded memory to enhance yield," *IEEE Design & Test of Computers*, vol. 18, no. 03, pp. 28–39, 2001.

[10] J. T. Chen, J. Rajski, J. Khare, O. Kebichi, and W. Maly, "Enabling embedded memory diagnosis via test response compression," in *Proceedings 19th IEEE VLSI Test Symposium. VTS 2001*, pp. 292–298, IEEE, 2001.

[11] I. Schanstra, D. Lukita, A. J. van de Goor, K. Veelenturf, and P. J. van Wijnen, "Semiconductor manufacturing process monitoring using built-in self-test for embedded memories," in *Proceedings International Test Conference 1998 (IEEE Cat. No. 98CH36270)*, pp. 872–881, IEEE, 1998.

[12] J.-F. Li and C.-W. Wu, "Memory fault diagnosis by syndrome compression," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, pp. 97–101, IEEE, 2001.

[13] C.-S. Hou, J.-F. Li, and T.-J. Fu, "A bist scheme with the ability of diagnostic data compression for rams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 2020–2024, 2014.

[14] A. J. Van de Goor, *Testing semiconductor memories: theory and practice*. John Wiley & Sons, Inc., 1991.

[15] G. Mrugalski, A. Pogiel, N. Mukherjee, J. Rajski, J. Tyszer, and P. Urbanek, "Fault diagnosis in memory bist environment with non-march tests," in *2011 Asian Test Symposium*, pp. 419–424, 2011.

[16] J. T. Chen, J. Khare, K. Walker, S. Shaikh, J. Rajski, and W. Maly, "Test response compression and bitmap encoding for embedded memories in manufacturing process monitoring," in *Proceedings International Test Conference 2001 (Cat. No. 01CH37260)*, pp. 258–267, IEEE, 2001.