Twin ECC: A Data Duplication Based ECC for Strong DRAM Error Resilience

Hyeong Kon Bae[1], Myung Jae Chung[1], Young-Ho Gong[2], and Sung Woo Chung[1]
[1]Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea
[2]School of Software, Soongsil University, Seoul 06978, South Korea
E-mail: {iqoong, qa7028, swchung}@korea.ac.kr; yhgong@ssu.ac.kr

***Abstract*—With the continuous scaling of process technology, DRAM reliability has become a critical challenge in modern memory systems. Currently, DRAM memory systems for servers employ ECC DIMMs with a single error correction and double error detection (SECDED) code. However, the SECDED code is insufficient to ensure DRAM reliability since memory systems become more susceptible to errors. Though various studies have proposed multi-bit correctable ECC schemes, such ECC schemes cause performance and/or storage overhead. To minimize performance degradation while providing strong error resilience, in this paper, we propose *Twin ECC*, a low-cost memory protection scheme through data duplication. In a 512-bit data, Twin ECC duplicates meaningful data into meaningless zeros. Since '1'→'0' error pattern is dominant in DRAM cells, Twin ECC provides strong error resilience by performing bitwise OR operations between the original meaningful data and duplicated data. After the bitwise OR operations, Twin ECC adopts the SECDED code for further enhancing data protection. Our evaluations show that Twin ECC reduces the system failure probability by average 64.8%, 56.9%, and 49.5%, when the portion of '1'→'0' error is 100%, 90%, and 80%, respectively, while causing only 0.7% performance overhead and no storage overhead compared to the baseline ECC DIMM with SECDED code.**

Keywords—DRAM reliability, data duplication, error correction code, bitwise operation

I. INTRODUCTION

The density and capacity of DRAMs have grown rapidly with the continued scaling of process technology. Though the scaling to smaller technology nodes enables higher density of DRAMs, it causes reliability challenges [5][8]. To ensure the DRAM reliability, error correction code (ECC) is widely exploited for modern memory systems. In general, recent DRAM memory systems for servers employ ECC DIMMs (Dual In-line Memory Modules) with a single error correction and double error detection (SECDED) code, which protects 64-bit data with 8-bit parity. However, as the error rate increases due to the technology scaling, several studies have proposed multi-bit correctable ECC schemes that provide stronger error correction capability than the SECDED code [4][6][7][10]. CARE adopts a 6-bit correctable Bose-Chaudhuri-Hocquenghem (BCH) code with memory page retirement techniques to prevent initially correctable errors from developing into uncorrectable errors later [4]. Stealth ECC has a 3-bit correctable BCH code in the meaningless part of the narrow-width value to protect the meaningful part from errors [10]; though Stealth ECC adopts a single stronger BCH code for narrow-width values only, we would like to leverage more meaningless parts more efficiently. AMD Chipkill exploits an 8-bit symbol Reed-Solomon (RS) code to correct any single symbol error from eighteen DRAM chips [18]. However, previously proposed ECC schemes incur substantial performance and/or storage overhead compared to the SECDED code.

One important observation not considered in previous ECC schemes is the asymmetric error behavior ('1'→'0' error pattern is dominant in true-cell regions[1]). A retention error occurs when a DRAM cell loses the charge before a refresh operation. A disturbance error occurs when the repetitive accesses lead a DRAM cell to lose the charge, such as Row Hammer [8][17]. Thus, it is natural that '1'→'0' error pattern is dominant. A radiation induced error is caused by alpha particles and cosmic neutrons. When the radiation induced error occurs in DRAM cells, it is reported that '1'→'0' error pattern is dominant [11][14][16]. However, when the radiation induced error occurs on the other circuits such as sense amplifiers and registers, it has the symmetric error behavior [11][14][16]. In other words, only except the case of the radiation induced error on the other circuits, '1'→'0' error pattern is dominant in the true-cell region [8][9][11][14][16][17]. Note some studies have reported that DRAM error behavior was not asymmetric, since they evaluated the true-cell and anti-cell regions together.

In this paper, we propose *Twin ECC*, a low-cost memory protection scheme providing strong error correction capability through data duplication. By exploiting the asymmetric error behavior of DRAM cells, Twin ECC improves the error correction capability with the bitwise OR operations between the original meaningful data and duplicated data; the result of the bitwise OR operations recovers all the '1'→'0' errors only if any '1'→'0' error does not occur in the same bit position between the original and duplicated data. To duplicate data without storage overhead in an ECC DIMM, Twin ECC classifies 64-bit data into four types: zero, narrow-width, same[2], and full-width values; since the zero value is a subset of the same value and narrow-width value, we classify them as mutually exclusive. In a 512-bit data, Twin ECC exploits meaningless zeros (zero values and upper bits of 32-bit narrow-width values) as a redundant space for meaningful data. Twin ECC also adopts the BCH (137,128,1) to correct any 1-bit error. Additionally, Twin ECC stores data with an interleaved manner to DRAM chips. The key contributions of this paper are as follows:

- Based on the asymmetric error behavior of DRAM cells, we propose *Twin ECC*, a strong error correction scheme through data duplication and ECCs.
- To duplicate data without additional storage overhead in an ECC DIMM, Twin ECC exploits meaningless zeros accounting for a considerable portion in DRAM.
- With bitwise OR operations instead of strong ECCs, Twin ECC achieves a short decoding latency leading to much less performance overhead compared to the previously proposed strong ECC schemes.

[1] In this paper, we focus on true-cell regions for simplicity.
[2] We define same value as a 64-bit value in which the upper 32-bit value is same as the lower 32-bit value.

978-3-9819263-7-8/DATE23/© 2023 EDAA

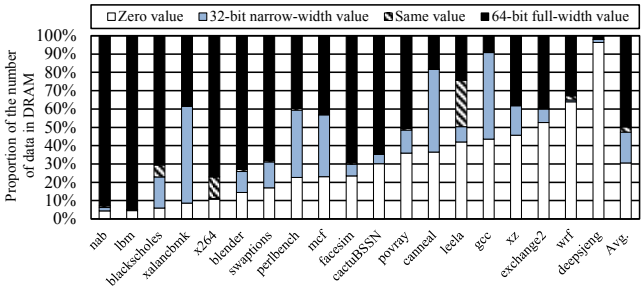


Fig. 1. Proportion of 64-bit data types in DRAM.

II. Background and Motivation

To protect memory systems against errors, various studies have employed multi-bit correctable BCH codes. Note the encoding process of BCH(n,k,t) converts k-bit data into n-bit codeword by exploiting (n-k)-bit parity for t-bit error correction and (t+1)-bit error detection. Typically, there is a trade-off among decoding latency, storage overhead, and error correction capability for BCH codes. For example, for 64-bit data, a 1-bit correctable BCH(72,64,1) requires 3 cycles of decoding latency with 8-bit parity (storage overhead), while a 3-bit correctable BCH(86,64,3) requires 10 cycles of decoding latency with 22-bit parity, which is calculated by Strukov's model based on the 22nm technology nodes [15]. Thus, ECC schemes employing strong BCH codes (i.e., multi-bit correctable BCH codes) increase the error correction strength at the expense of performance and storage overhead.

To provide strong error resilience while mitigating the performance and storage overhead, we exploit meaningless zeros of narrow-width data values. According to [10], 32-bit narrow-width values (i.e., a 64-bit data consists of 32-bit zero values with 32-bit non zero values) in DRAM account for more than 40%, on average. We investigate the proportion of 64-bit data types for duplication (i.e., zero, narrow-width, and same values) in DRAM for SPEC CPU 2017 [3] and PARSEC [1] workloads[3]. As shown in Fig. 1, the proportion of 64-bit data types for duplication in DRAM accounts for 50.2% (30.6/16.8/2.8% for zero/narrow-width/same values, respectively), on average. Therefore, it is efficient to enhance DRAM reliability through duplication.

III. Twin ECC: Data Duplication Based ECC

In this section, we first describe a brief overview of our proposed Twin ECC, followed by a detailed description of the encoding and decoding process.

A. Overview

Fig. 2 depicts the hardware components of Twin ECC in the memory controller. As shown in the top of Fig. 2, on a memory write operation (encoding), Twin ECC requires a data duplicator, ECC encoder, flag encoder, and bitwise interleaver. In case of a memory write operation, the data duplicator reorganizes the 512-bit data by duplicating 64-bit full-width values and 32-bit narrow-width values into zero values and upper 32 bits of 32-bit narrow-width values, respectively, if possible; for the same values, we consider the upper 32-bit same values as duplicated data of the lower 32-bit same values. The duplicated data is exploited to improve error correction capability, based on the bitwise OR operations with the original data. In addition to the data

[3] We only consider the working set of each workload, not the entire memory space.

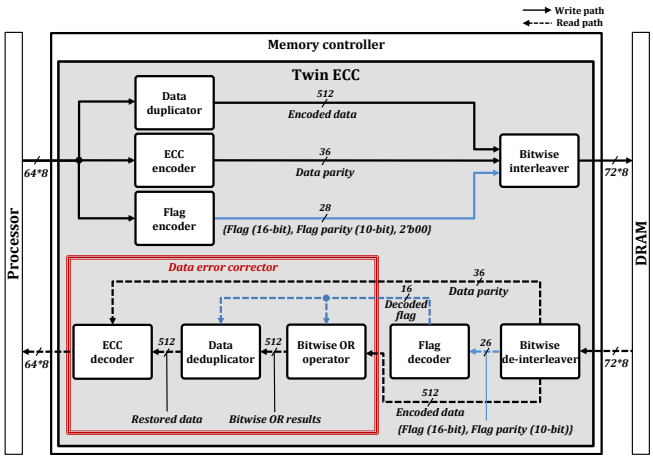


Fig. 2. Overview of Twin ECC.

duplication, the ECC encoder applies the BCH(137,128,1) to each 128-bit of the 512-bit data. Since Twin ECC applies data duplication depending on the data type, it requires flag bits to distinguish data types. Thus, the flag encoder generates a 2-bit flag for each 64-bit data (totally 16-bit flag for a 512-bit data); to prevent error in the flag bits, it applies the BCH(13,8,1) to each 8-bit of the 16-bit flag. Lastly, to further improve DRAM reliability, the bitwise interleaver stores the encoded data, data parity, flag, and flag parity in a bitwise interleaved manner across DRAM chips.

As shown in the bottom of Fig. 2, on a memory read operation (decoding), Twin ECC requires a bitwise de-interleaver, flag decoder, bitwise OR operator, data deduplicator, and ECC decoder. When a memory read operation occurs, the bitwise de-interleaver loads the encoded data, data parity, flag, and flag parity from DRAM chips. Then, the flag decoder accurately decodes the 16-bit flag by exploiting the 10-bit flag parity. Based on the 16-bit decoded flag, the bitwise OR operator corrects '1'→'0' error(s) by performing bitwise OR operations between the original meaningful data and duplicated data in the 512-bit encoded data, if duplicated. After the bitwise OR operations, the bitwise OR results are stored in the original meaningful data locations in a 512-bit restored data, while the original meaningless zeros locations are filled with zeros by the data deduplicator to protect '0'→'1' error(s); the bitwise OR results for the same values are stored in both the locations of the original upper 32-bit same values and the original lower 32-bit same values in a 512-bit restored data. Then, the ECC decoder checks the 512-bit restored data by exploiting the 36-bit data parity. Lastly, the 512-bit decoded data (i.e., original data) is transferred to the processor.

B. Encoding Process

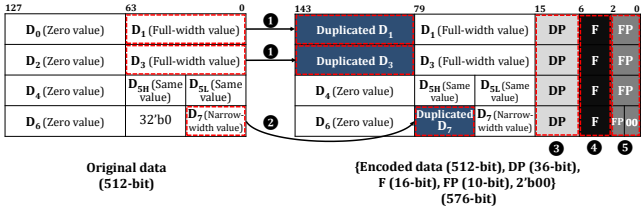
Since we classify a 64-bit data into four types, there needs to be a 2-bit flag to identify them. Thus, a 16-bit (=2-bit * 8) flag is needed for each 512-bit data. To make room for the 16-bit flag without additional storage overhead in an ECC DIMM, we adopt the BCH(137,128,1) instead of baseline BCH(72,64,1); though BCH(137,128,1) causes only 2.3% reliability degradation compared to the baseline BCH(72,64,1) [10], Twin ECC further improves the reliability through duplication. While the baseline BCH(72,64,1) for the 512-bit data requires 64-bit (=8-bit * 8) parity, the BCH(137,128,1) for the 512-bit data requires only 36-bit (=9-bit * 4) parity. With the BCH(137,128,1), it is possible to free up 28-bit (=64-bit – 36-bit) space in the 576-bit (corresponding to eight memory bursts). Accordingly, we store the 16-bit flag and 10-bit flag parity in this 28-bit freed

TABLE I. FLAG VALUES DEPENDING ON THE DATA TYPES FOR DATA CLASSIFICATION

Data type	Flag
Full-width value	00
Zero value	01
Narrow-width value	10
Same value	11

space, and fill the remaining 2-bit space with 2'b00; 10-bit flag parity is generated by applying the BCH(13,8,1) to each 8-bit of the 16-bit flag. As '1'→'0' error pattern is dominant in DRAM cells, it would be better for reliability to store '0's rather than '1's as many as possible. Thus, considering the portion of data types shown in Fig. 1, we configure the flag bits such that they are stored as '0's in DRAM as many as possible. As described in Table I, the most prevalent data type (i.e., full-width value) has 2'b00 for its flag, while the least prevalent data type (i.e., same value) has 2'b11 for its flag.

Fig. 3 illustrates the encoding process of our proposed Twin ECC with an example. To protect the 512-bit original data from '1'→'0' error(s), the data duplicator creates a 512-bit encoded data by duplicating meaningful data into meaningless zeros in the 512-bit original data, if possible. Specifically, 64-bit zero values and upper 32 bits of 32-bit narrow-width values are exploited as a redundant space for 64-bit full-width values and 32-bit narrow-width values, respectively; for same values, we consider the upper 32-bit same values as duplicated data of the lower 32-bit same values. To describe the duplication process for full-width values in detail, we depict how full-width values are duplicated to the redundant space depending on the number of zero values and full-width values in the 512-bit original data, as shown in Fig. 4. Within the 512-bit original data, the locations of zero values and full-width values are identified in the ascending address order; the earlier the values (zero values and full-width values) are identified, the higher the priority exploited for duplication. When there is no zero value or no full-width value in the 512-bit original data, the duplication of full-width values is not possible. When the number of zero values is less than the number of full-width values, the full-width values are duplicated to the zero values as many as possible, depending on the ascending address order. In this case, one or more full-width values cannot be duplicated. When the number of zero values is greater than or equal to the number of full-width values, the full-width values are duplicated to the zero values, depending on the ascending address order. In this case, one or more zero values are not exploited as a redundant space. For example, as shown in Fig. 3, there are four zero values, two full-width values, one narrow-width value, and one same value in the 512-bit original data. To duplicate the full-width values into the zero values, the locations of zero values and full-width values are identified in the ascending address order (i.e., D_0 to D_7), respectively. Since the number of zero values is greater than the number of full-width values, the two initially identified zero values (i.e., D_0 and D_2) are exploited as a redundant space for the two initially identified full-width values (i.e., D_1 and D_3) (❶); the remaining two zero values (i.e., D_4 and D_6) are not exploited. In case of the narrow-width value, the 32-bit narrow-width value (i.e., D_7) is duplicated to the upper 32 bits (i.e., meaningless zeros) (❷). In case of the same value, we consider the upper 32-bit same value (i.e., D_{5H}) as duplicated data of the lower 32-bit same value (i.e., D_{5L}). In addition, the ECC parity generator creates a 36-bit data parity by applying the BCH(137,128,1) to each 128-bit of the 512-bit original data (❸). Since we



*D, DP, F, and FP denote data, data parity, flag, and flag parity, respectively.

Fig. 3. Encoding process of our proposed Twin ECC.

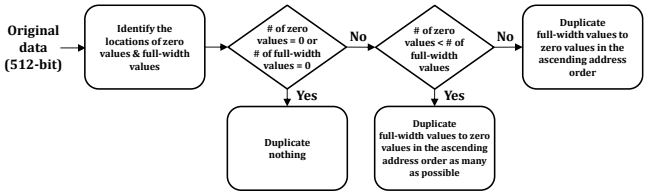


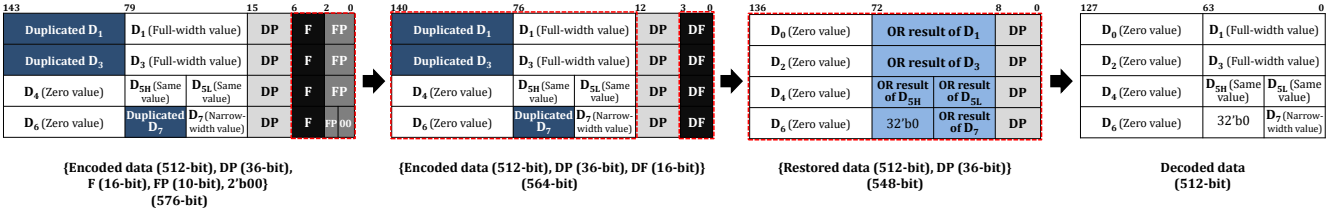
Fig. 4. Duplication process for full-width values in the data duplicator.

configure the 2-bit flag for each 64-bit data type, the flag encoder sets a 16-bit flag for the 512-bit original data (❹). Though Twin ECC improves DRAM reliability through data duplication and BCH(137,128,1), in case of errors occur in the flag bits it misclassifies the data types which may result in system failure. To resolve this problem, each 8-bit of the 16-bit flag is protected by the BCH(13,8,1) in the flag encoder (❺).

Lastly, to further improve DRAM reliability, the bitwise interleaver stores the encoded data, data parity, flag, and flag parity in a bitwise interleaved manner across eighteen x4 DRAM chips, which is deployed in [10]. The bitwise interleaved mapping ensures that two bits in the same bit position between the original and duplicated data are not stored in the same DRAM chip. When all the meaningful data are duplicated in the 512-bit encoded data, Twin ECC is tolerable to any single chip failure based on the bitwise OR operation between the original and duplicated data. In case of 36-bit data parity, each 9-bit of 36-bit is stored into different DRAM chips, so that any single chip failure is recovered by BCH(137,128,1). In case of 16-bit flag and 10-bit flag parity, each 13-bit (8-bit flag+5-bit flag parity) is scattered into different DRAM chips and thus any single chip failure is tolerable by BCH(13,8,1). However, when using the conventional mapping method, four consecutive bits in the 72-bit (one memory burst) including encoded data, data parity, flag and flag parity are stored in a single DRAM chip. With the conventional mapping method, a single chip failure results in 4-bit error in the encoded data, data parity, flag, and/or flag parity of the consecutive 72-bit, which is uncorrectable by BCH(137,128,1) and/or BCH(13,8,1), leading to system failure.

C. *Decoding Process*

Fig. 5 illustrates the decoding process of our proposed Twin ECC with an example. To read data from main memory, the bitwise de-interleaver loads the encoded data, data parity, flag, and flag parity from DRAM chips. Then, as shown in the first process (i.e., leftmost in Fig. 5), each 8-bit of the 16-bit flag is accurately decoded by the BCH(13,8,1) in the flag decoder. Based on the 16-bit decoded flag, the locations of the original meaningful data and the duplicated data are identified within the 512-bit encoded data. Then, the bitwise OR operator corrects '1'→'0' error(s) by performing bitwise OR operations between the original meaningful data and duplicated data, if duplicated. After the bitwise OR operations, the bitwise OR results are stored in the original meaningful data locations in a 512-bit restored data, while



*D, DP, F, FP, and DF denote data, data parity, flag, flag parity, and decoded flag, respectively.

Fig. 5. Decoding process of our proposed Twin ECC. Red dotted boxes indicate the data exploited in the next process.

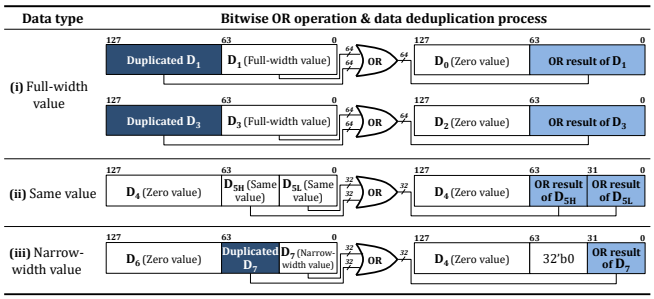


Fig. 6. Bitwise OR operations and data deduplication process for each data type.

the original meaningless zeros locations are filled with zeros by the data deduplicator, as shown in the second process in Fig. 5; for same values, the bitwise OR results are stored in both the locations of the original upper 32-bit same values and the original lower 32-bit same values in a 512-bit restored data. To describe the second process in detail, we depict the bitwise OR operations and data deduplication process for each data type in the 512-bit encoded data, as shown in Fig. 6. For the (i) full-width values, 64-bit bitwise OR operations are performed between 64-bit full-width values (i.e., D_1 and D_3) and 64-bit duplicated data (i.e., duplicated D_1 and duplicated D_3). After the bitwise OR operations, the 64-bit bitwise OR results are stored in the locations of the original full-width values in the 512-bit restored data, while the locations of the original 64-bit zero values (i.e., D_0 and D_2) are filled with zeros. For the (ii) same value, 32-bit bitwise OR operations are performed between the upper 32-bit same value (i.e., D_{5H}) and the lower 32-bit same value (i.e., D_{5L}). After the bitwise OR operations, the 32-bit bitwise OR result is stored in both the locations of the original upper 32-bit same value and the original lower 32-bit same value in the 512-bit restored data. For the (iii) narrow-width value, 32-bit bitwise OR operations are performed between 32-bit narrow-width value (i.e., D_7) and 32-bit duplicated data (i.e., duplicated D_7). After the bitwise OR operations, the 32-bit bitwise OR result is stored in the location of the original 32-bit narrow-width value in the 512-bit restored data, while the upper 32-bit location of the original 32-bit narrow-width value is filled with zeros. Meanwhile, when '1'→'0' errors occur in the same bit position between the original and duplicated data, or '0'→'1' errors occur (which is expected to be very rare), they cannot be corrected by the bitwise OR operations. In this case, each 128-bit of the 512-bit data is corrected by the BCH(137,128,1) in the ECC decoder, as shown in the last process (i.e., rightmost in Fig. 5).

IV. Evaluation

A. Experimental Environment

We evaluate Twin ECC in terms of DRAM reliability, performance, and area/power overhead, compared to the baseline (ECC DIMM with BCH(72,64,1)), BCH(573,512,6), and Stealth ECC (which adopts bitwise interleaving) [10]. We conduct our evaluations with nineteen workloads from SPEC CPU 2017 [3] and PARSEC [1] benchmark suites. Specifically, in case of the single-threaded SPEC benchmark suite, we run each workload with a single thread. In case of the PARSEC multi-threaded benchmark suite, we run each workload with four threads to consider the impact of atomic memory operations and thread synchronization. For each simulation, we fast-forward the first 10 billion instructions and then execute 1 billion instructions. We use Faultsim [13], a configurable memory-reliability simulator, to compare the reliability based on real-world failure statistics for DRAM devices. Considering the asymmetric error behavior of true-cell regions, we set the portion of '1'→'0' DRAM error as 100%, 90%, and 80%. To evaluate the system failure probability (i.e., the probability of an uncorrectable error in the system), we perform Monte-Carlo simulations for a 7-year period with 10 million iterations. For performance evaluation, we calculate the decoding latency of BCH codes by exploiting Strukov's model [15]. Based on the Strukov's model, the decoding latency of BCH codes employed by Twin ECC is 6 cycles. In the case of Twin ECC, we also extract the latency of additional hardware components (e.g., bitwise de-interleaver, bitwise operator, data deduplicator, etc.) by implementing the hardware components in Verilog HDL and then synthesizing them at 3GHz frequency using the Synopsys Design Compiler with SAED 14nm FinFET process technology [12]. According to the synthesis results, one cycle is enough to execute the bitwise de-interleaver, bitwise OR operator, and data deduplicator (explained in Section III-C). We reflect the estimated latency (7 cycles = 6 + 1) to the gem5 simulator [2]. The detailed system parameters for performance evaluation are described in Table II. In addition, we analyze the area and power consumption of Twin ECC based on the synthesis result.

B. Reliability Evaluation

Fig. 7 compares the system failure probability across nineteen workloads. When the portion of '1'→'0' error is 100/90/80%, Twin ECC reduces the average system failure probability by 64.8/56.9/49.5%, 50.3/39.2/28.8%, and 31.9/16.7/2.5% compared to the baseline, BCH(573,512,6), and Stealth ECC [10], respectively, due to the following reasons. First, Twin ECC provides multi-bit (up to 64-bit) correction capability for zero, narrow-width, and same values,

TABLE II. Configuration Parameters

Parameter	Configuration
Processor	4 cores; out-of-order; x86; 3GHz
Cache	L1D/L1I: Private; 32KB; 8-way; 64B block L2: Shared; 256KB; 16-way; 64B block
Memory controller	Decoding latency including flag decoding and bitwise interleaving (cycle): Baseline* (3), BCH(573,512,6) (24), Stealth ECC (10), Twin ECC (7)
Main memory	DDR4-2400; x4 bus-width; 18 chips; 16GB

*Note our baseline is ECC DIMM with BCH(72,64,1).

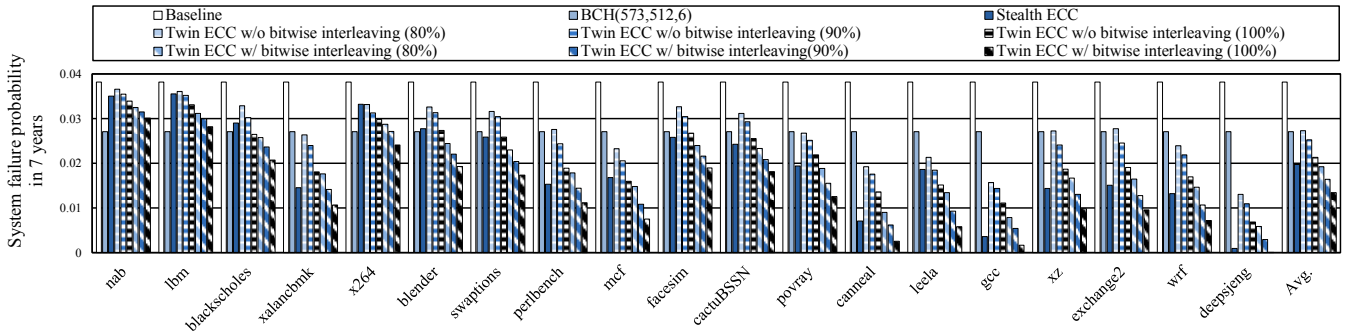


Fig. 7. System failure probability depending on ECC schemes.

while the baseline provides only 1-bit correction capability for each 64-bit data. Second, Twin ECC provides stronger error resilience than BCH(573,512,6) and Stealth ECC in most workloads, since Twin ECC is capable of correcting 20-bit per a 512-bit data, on average, in our simulation; BCH(573,512,6) and Stealth ECC correct average 6-bit and 14-bit per a 512-bit data, respectively. Hence, as the proportion of data types for duplication (i.e., zero, narrow-width, and same values) increases, Twin ECC further reduces the system failure probability. For example, as shown in Fig. 7, in the case of *deepsjeng*, *gcc*, *leela*, and *canneal,* Twin ECC leads to much lower system failure probability than the other ECC schemes, since the proportion of data types for duplication is higher (more than 70%). However, as the proportion of data types for duplication decreases, Twin ECC gets less efficient for system failure reduction than BCH(573,512,6), leading to higher system failure probability. For example, in the case of *nab* and *lbm* (the two leftmost workloads in Fig. 7), Twin ECC still shows lower system failure probability than baseline and Stealth ECC, but higher system failure probability than BCH(573,512,6). However, BCH(573,512,6) has 3.4x longer decoding latency compared to Twin ECC, causing worse system performance (will be described in the next subsection).

To verify the effectiveness of Twin ECC, we evaluate average correction coverage for each data type across nineteen workloads shown in Table III; the correction coverage is ratio of error tolerance (the higher the better) for each data type in our simulation. In case of the zero and narrow-width values, Stealth ECC shows high correction coverage, since it adopts a strong BCH code with bitwise interleaved mapping method. Twin ECC also shows higher correction coverage through the bitwise OR operations and BCH(137,128,1). In case of the same and full-width values, since Stealth ECC employs the baseline or BCH(137,128,1), it shows lower correction coverage than the baseline. On the other hand, Twin ECC provides high correction coverage for the same values by performing bitwise OR operations between upper 32-bit same values and the lower 32-bit same values. Moreover, for full-width values, Twin ECC shows higher correction coverage compared to the other ECC schemes by exploiting a considerable portion of zero values as a redundant space. Accordingly, though Twin ECC adopts the BCH(137,128,1) with lower correction coverage than the baseline, it provides higher reliability by exploiting the bitwise OR operations with data duplication.

C. Performance Evaluation

Fig. 8 shows the normalized execution time of Twin ECC and the other ECC schemes across nineteen workloads. Twin ECC shows the performance overhead by only 0.7%, on average, compared to the baseline. Typically, the decoding latency rather than the encoding latency more affects the system performance. In the decoding process, Twin ECC with the bitwise OR operation, BCH(137,128,1), and BCH(13,8,1) is much simpler than the strong ECCs. As shown in Table II, Twin ECC needs 7 cycles for decoding latency for BCH codes and additional hardware components. In contrast, BCH(573,512,6) and Stealth ECC needs 24 cycles and 10 cycles for decoding latency, which incurs performance overhead by average 4.0% and 1.3%, respectively.

TABLE III. COMPARISON OF CORRECTION COVERAGE FOR EACH DATA TYPE

ECC scheme	*Baseline*	*BCH (573,512,6)*	*Stealth ECC [10]*	*Twin ECC (80%)*	*Twin ECC (90%)*	*Twin ECC (100%)*
Zero value	48.41%	59.65%	99.58%	89.71%	94.73%	99.92%
Narrow-width value	48.17%	59.29%	99.5%	90.32%	95.76%	99.95%
Same value	48.08%	59.27%	46.82%	89.13%	94.21%	99.79%
Full-width value	47.63%	59.18%	46.36%	58.29%	59.63%	61.03%

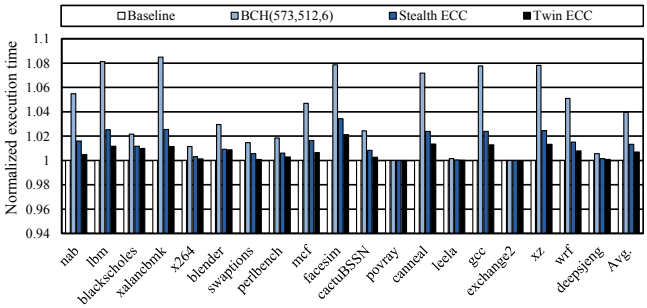


Fig. 8. Normalized execution time depending on ECC schemes.

D. Area and Power Analysis

As shown in Table IV, the area and power consumption of Twin ECC are 0.017mm^2 and 2.38mW, respectively. Though the area of Twin ECC is larger than the area of baseline and Stealth ECC, the hardware components of Twin ECC incur negligible area overhead (i.e., 0.25%) compared to the memory controller area (i.e., 6.9mm^2) of a state-of-the-art server CPU [19]. Furthermore, the power consumption of Twin ECC is small compared to the thermal design power (i.e., 205W) of the state-of-the-art server CPU [19].

V. RELATED WORK

Various previous studies have presented ECC schemes to enhance DRAM reliability. Among them, we describe the previous studies that provided Chipkill-level or near Chipkill-level ECC schemes, as shown in Table V. Kim et al. proposed Bamboo ECC which employs vertical RS code to protect any single DRAM chip failure at cache line granularity [6]. Kim et al. also proposed Frugal ECC, a compression-based ECC scheme, which is tolerable to a

TABLE IV. AREA AND POWER COMPARISON

ECC scheme	*Baseline*	*BCH(573,512,6)*	*Stealth ECC [10]*	*Twin ECC (This work)*
Area	0.001mm^2	0.08mm^2	0.01mm^2	0.017mm^2
Power*	0.58mW	9.26mW	1.63mW	2.38mW

*The sum of dynamic power and leakage power

single DRAM chip error by exploiting RS code [7]. Though Bamboo ECC [6] and Frugal ECC [7] provide DRAM chip error resilience, they incur performance overhead by average 19.3% and 21.4%, respectively. In addition, Frugal ECC causes storage overhead by 12.0%, since it exploits extra memory space to store parity bits in case of compression failure. Chen et al. presented CARE, which exploits a 6-bit correctable BCH code with operating system (OS) support [4]. CARE tries to mitigate the storage overhead in DRAM by employing a cache-like structure (i.e., ECC cache) to store the parity bits of the BCH code. Based on the observation that initially correctable errors are converted into uncorrectable errors later, CARE corrects the errors through the ECC cache, and then retires the memory page. Though CARE enhances DRAM reliability thanks to the page retirement techniques, it causes average 10.0% performance overhead in error-dominant cases; in nearly error-free cases, CARE incurs about 1.0% performance overhead. Lee et al. proposed Stealth ECC, a data-width aware adaptive ECC scheme, which provides near Chipkill-level reliability by employing a narrow-width value feature [10]. Stealth ECC applies the BCH(51,32,3) to the meaningful part of narrow-width values by exploiting the meaningless zeros as a storage space for the parity bits. However, since Stealth ECC adopts weak BCH codes (i.e., BCH(72,64,1) or BCH(137,128,1)) for full-width values, it may have worse reliability than the baseline (i.e., BCH(72,64,1)) for full-width values. On the other hand, Twin ECC provides strong protection for the full-width values by exploiting the bitwise OR operations with data duplication as well as BCH(137,128,1). Moreover, since the next-generation DRAM (e.g., DDR5 [20]) supports a mode that increases the memory burst length to 1024-bit rather than 512-bit, it is possible to duplicate more full-width values into the redundant space.

VI. CONCLUSION

We propose Twin ECC, a low-cost memory protection scheme which employs data duplication and ECCs to provide strong error resilience. To duplicate data without any storage overhead in an ECC DIMM, Twin ECC exploits meaningless zeros as a redundant space for meaningful data, within a 512-bit data. Since '1'→'0' error pattern is dominant in DRAM cells, Twin ECC performs bitwise OR operations between the original meaningful data and duplicated data. After the bitwise OR operations, Twin ECC adopts the BCH(137,128,1) for further error resilience. Consequently, when the portion of '1'→'0' error is 100/90/80%, Twin ECC enhances the DRAM reliability by (64.8/56.9/49.5%), (50.3/39.2/28.8%), and (31.9/16.7/2.5%) compared to the baseline BCH(72,64,1), BCH(573,512,6), and Stealth ECC, respectively. Furthermore, Twin ECC incurs negligible performance overhead (0.7%, on average) and no storage overhead, compared to the baseline ECC DIMM with BCH(72,64,1). Though our proposed Twin ECC is based on true-cell regions in this paper, Twin ECC is applicable to anti-cell regions as well. Since '0'→'1' error pattern is dominant in anti-cell regions, it is possible to provide strong error resilience by performing bitwise AND operations between the original meaningful data and duplicated data.

TABLE V. COMPARISON BETWEEN RELATED WORK WITH TWIN ECC

ECC scheme	Chipkill-level		Near Chipkill-level		
	Bamboo ECC [6]	*Frugal ECC [7]*	*CARE [4]*	*Stealth ECC [10]*	*Twin ECC (This work)*
Storage overhead (vs. BCH(72,64,1))	0%	12.0%	0%[a]	0%	0%
Avg. perf. overhead (vs. BCH(72,64,1))	19.3%	21.4%	~10.0%[b]	1.3%	0.7%
OS support	No	No	Needed	No	No

[a] It additionally requires a 56KB ECC cache per 8GB DRAM.
[b] It depends on whether the memory page is retired or not.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1A2C2003500), Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00441-001, Memory-Centric Architecture Using the Reconfigurable PIM Devices), and Samsung Electronics. We would like to thank Prof. Jung Ho Ahn for providing helpful insights. Sung Woo Chung and Young-Ho Gong are the co-corresponding authors of this paper.

REFERENCES

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.
[2] N. Binkert et al., "The gem5 simulator," ACM SIGARCH Computer Architecture News, vol. 39, no. 2, pp. 1-8, 2011.
[3] J. Bucek, K.-D. Lange, and J. Kistowski, "SPEC CPU2017: Next-generation compute benchmark," in *ICPE*, 2018.
[4] J. Chen et al., "CARE: Coordinated augmentation for elastic resilience on DRAM errors in data centers," in *HPCA*, 2021.
[5] A. Fakhzadehgan et al., "Safeguard: Reducing the security risk from Row-Hammer via low-cost integrity protection," in *HPCA*, 2022.
[6] J. Kim, M. Sullivan, and M. Erez, "Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory," in *HPCA*, 2015.
[7] J. Kim, M. Sullivan, S.-L. Gong, and M. Erez, "Frugal ECC: Efficient and versatile memory error protection through fine-grained compression," in *SC*, 2015.
[8] Y. Kim et al., "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ISCA*, 2014.
[9] K. Kraft et al., "Improving the error behavior of DRAM by exploiting its Z-channel property," in *DATE*, 2018.
[10] Y. S. Lee, G. Koo, Y.-H. Gong, and S. W. Chung, "Stealth ECC: A data-width aware adaptive ECC scheme for DRAM error resilience," in *DATE*, 2022.
[11] S. Liu et al., "Exploiting asymmetry in eDRAM errors for redundancy-free error-tolerant design," *IEEE Transactions on Emerging Topics in Computing,* vol. 9, no. 4, pp. 2064-2075, 2021.
[12] V. Melikyan et al., "14nm educational design kit: Capabilities deployment and future," *Small Systems Simulation Symposium*, 2018.
[13] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "Faultsim: A fast, configurable memory-reliability simulator for conventional and 3D-stacked systems," *ACM Transactions on Architecture and Optimization*, vol. 12, no. 4, pp. 1-24, 2016.
[14] B. Narasimham and W. K. Luk, "A multi-bit error detection scheme for DRAM using partial sums with parallel counters," in *IEEE International Symposium on Reliability Physics*, 2008.
[15] D. Strukov, "The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories," *Fourtieth Asilomar Conference on Signals, Systems and Computers*, 2006.
[16] S. Wang et al., "Content aware refresh: Exploiting the asymmetry of DRAM retention errors to reduce the refresh frequency of less vulnerable data," *IEEE Transactions on Computers*, vol. 68, no. 3, pp.362-374, 2019.
[17] X.-C. Wu et al., "Protecting page tables from RowHammer attacks using monotonic pointers in DRAM true-cells," in *ASPLOS*, 2019.
[18] Advanced Micro Devices, "BIOS and kernel developer's guide (BKDG) for AMD family 15h models 00h-0fh processors," 2013.
[19] Intel, "Intel® Xeon® Gold 6338 Processor," 2021. [Available]: https://ark.intel.com/content/www/us/en/ark/products/212285/intel-xeon-gold-6338-processor-48m-cache-2-00-ghz.html.
[20] JEDEC, "DDR5 SDRAM," 2020.