Fault-tolerant architecture for Cache: Summaries, Assessments and Trends

Xuru Wang¹,², Xin Gao¹,², Zongnan Liang³, Jiawei Nian³,⁴ and Hongjin Liu¹,²*

¹ Beijing Institute of Control Engineering, Beijing, 100090, China
² Beijing SunWise Space Technology Ltd, Beijing 100090, China
³ School of computer science and technology, Xidian University, Xi’an, Shanxi, 710071, China.
⁴Corresponding author’s e-mail: wangxr0321@cast.cn

Abstract: Fault-tolerant design of cache is a key aspect of highly reliable processor design. In this paper, based on the key metrics in Cache architecture design: reliability, power consumption, latency and area, we divided the related research into two categories: one is to maximize reliability with guaranteed latency, power consumption and area, the other is to minimize latency, power consumption and area loss while ensuring fault tolerance reliability. Based on the classification, by analyzing different studies of Data and Tag in Cache, this paper gives the characteristics of these methods and the future development trend.

1. Introduction
Accuracy of data and instructions is an important guarantee in computer processes. However, in some specific environments such as space environment, the correctness of processor execution is seriously harmed because of the exposure to extreme conditions like radiation [5][8][12]. In particular, single and multiple bit upsets can cause transient or permanent changes in data bits, which can lead to catastrophic failures. In the past decades, Single Event Upset (SEU) was considered as the main influencing factor for Cache reliability, but with the gradual reduction of the feature size of semiconductor devices, the chance of Multiple-Bits Upset (MBU) is increasing and needs the same attention in the design [15][17][21]. Therefore, in processor design, the reliability design of the Cache, the key part in data processing, should be mainly concerned. In the design of fault-tolerant computer architecture, four metrics should be focused: reliability, latency, power consumption, and area. Therefore, recent studies can be divided into the following two categories according to the indicators of concern: the first one is to maximize fault tolerance with guaranteed latency, power consumption and area, the other is to minimize latency, power consumption and area loss while ensuring fault tolerance reliability [22]. In this paper, we analyze and evaluate these two types of research, summarize their technical points, and give the development trend of fault-tolerant technology for Cache architecture.

The data in Cache is divided into Data and Tag, where the differences are the way they are stored. Therefore, the related research can be mainly divided into fault-tolerant protection for Data and fault-tolerant protection for Tag. This paper is organized as follows: The section 2 provides an overview of the basic fault-tolerance methods and their principles. In Section 3 and Section 4, the relevant researches on the protection of Data and Tag is analyzed, respectively. Section 4 summarizes the whole paper and gives the development trend of Cache fault-tolerance protection technology.
2. Methods and basic principles
The main form of fault-tolerant design for Cache architecture in common use today is redundant design, which is divided into three types: temporal redundancy, spatial redundancy, and information redundancy. Temporal redundancy means using the same hardware to repeat instruction operations to ensure reliability. However, this is a sacrifice of time for reliability, and the execution speed of each instruction will become slower, which is contradictory to the goal of pursuing high performance. Thus, temporal redundancy is rarely used in design. Spatial redundancy, on the other hand, uses multiple backups on hardware circuits, and instructions are executed in these circuits simultaneously to ensure reliability by comparison of results, but it will cause an increase in power consumption and area. Information redundancy uses additional EDC (Error Detection Code) and ECC (Error Correction Code) to protect the data, and the area power consumption is relatively smaller to spatial redundancy. However, the delay problem could arise due to the extra coding and decoding logic. The following are some classic and commonly used fault tolerance methods:

2.1. TMR (Triple Modular Redundancy)
Triple Modular Redundancy (TMR), for short, is one of the most commonly used spatial redundancy. Three same modules will perform the instruction simultaneously, taking the majority through voting system as correct output, which is often referred to as triple take two. Therefore, the correct output of system can be guaranteed unless the same error occurs in at least two modules., the error of the faulty module can be masked and the correct output of the system can be guaranteed. The structure is illustrated in figure 1. Since the three modules are independent of each other, the probability that two modules have errors simultaneously is quite small., so the reliability of the system can be greatly improved.

Figure 1. The structure of TMR

The reliability of each module is assumed to be equal and is denoted by $R_m$. The reliability of the table decider is denoted by $R_v$. Then the system reliability is:

$$R = (3R_m^2 - 2R_m^3) \cdot R_v$$

The key to this structure is how to implement the design of the three-take two, which is the voting machine.

2.2. Parity Check
Parity checking is a traditional method of information redundancy for fault tolerance [10]. There are two types of parity bits: even parity bits and odd parity bits. If the number of 1 bit in a given set of data bits is odd, then the even parity bit is set to 1, so that the total number of 1 bit is even. If the number of 1’s in a given set of data bits is even, then the odd parity bit is set to 1, making the total number of 1’s odd. Even parity is actually a special case of cyclic redundancy, where a 1-bit CRC is obtained by the polynomial $x^1$.

If an odd number of data bits including the parity bit are flipped, the parity bit will be changed, indicating an error during transmission. Therefore, the parity bit is an error detection code, but since there is no way to determine which bits are flipped, it cannot be error corrected. Once an error is detected, the received data have to be discarded and then transmission reruns.
2.3. **Hamming Code**

Hamming codes can be used to detect and correct single bit errors when a processor stores or moves data that may generate data bit errors. Because of the simplicity of Hamming codes, they are widely used in storage units [1][7].

Hamming code utilizes the concept of parity bits by adding some bits after the data bits, the validity of the data can be verified. By using more than one parity bit, Hamming Code not only verifies the validity of the data, but also indicates the error location in case of data errors.

The purpose of a Hamming code is to correct a single bit of error. It can be expressed as Hamming(n,r), where the information code has n bits and the Hamming code has r bits, the total code length is n + r bits. In order to detect an error in one of the n + r bits, the Hamming code must be able to represent at least n + r + 1 states, where n + r means that there is an error in one of the n + r bits, and the extra one means that the whole code is correct. Then the length of the Hamming code needs to satisfy the following relationship:

\[ 2^r > n + 1 + r \]

The encoding algorithm for the Hamming code is as follows.
1. fill the new code as 2^(r-1), (r ≥ 0) bits with zeros (i.e. parity bits)
2. fill the remaining bits of the new code with the source code in the original order
3. encode the rth check digit starts from then the 2^(r-1) bit of the new code, and for every 2^(r-1) bits of the iso-or calculated, jump 2^(r-1) bits, and then calculate the next set of 2^k bits of the iso-or, fill in 2^(r-1) bits. The rth check digit is then computed from the 2^k bits, (r = 1,2,3 ...th bit of the new code. Each time it needs to calculate the2^(r-1) iso-or, jump 2^(r-1) bits, then calculate the next set of 2^(r-1) bits iso-or, fill in the 2^(r-1) position.

If the checksum bits generated by the receiver and the received checksum bits are the same, the data is correct, otherwise they are different indicating an error.

3. **Fault-Tolerant Protection of Cache Data**

Since the method of spatial redundancy is mainly designed for fault tolerance through the method of backup, which is relatively simple and has rarely studies on. Therefore, the research on the fault-tolerant architecture of Cache data mainly focuses on information redundancy.

When designing information redundant architecture, we mainly focus on four metrics: reliability, power consumption, area and latency, but these four metrics interact with each other and need to be considered in a compromise. Therefore, the research related to information redundancy can be divided into two main classes:
1. Maximize fault tolerance reliability with guaranteed latency, power consumption and area;
2. Minimize latency, power consumption and area loss while ensuring fault tolerance reliability.

In the first group of studies, researchers primarily focused on reliability improvements, for example, improving the data detection and correction range of EDAC codes. This will inevitably result in complex logic circuits, increasing power consumption. The second group of studies primarily focuses on optimizing the overall performance of the processor after adding a fault-tolerant design. In these studies, the loss of power and latency is alleviated by adjusting the structure of the Cache read/write path and the read/write and replacement strategies.

3.1. **Maximize fault tolerance reliability with guaranteed latency, power consumption and area**

The Common approaches to information redundancy are Hamming code, RM code [2] (Reed Muller code), RS code [3] (Reed-Solomon code), BCH code [4] (Bose-Chaudhuri-Hocquenghem), etc.

Hamming code is simple in logic [1], easy to implement, and is the most commonly used EDAC code for Cache information redundancy. However, its weak error correction capability (only one bit error correction) cannot satisfy the growing reliability requirements, because the size of devices decreases and the probability of MBU gradually increases with the rapid development of semiconductors. RS, RM, BCH codes have more complex logic and higher error correction and
detection capability, for example, RM (2, 5) can achieve three error correction, but the complex circuit logic will lead to more delay and power loss. In a word, although the implementation of Hamming code is simple, its reliability is not high; while RS, RM, BCH code is more reliable, but its coding and decoding logic is complex. Therefore, in this part, relevant researches are mainly divided into two kinds: one is to optimize the coding and decoding logic to improve the error tolerance with the same amount of redundancy; the other is to use basic ECC and EDC (such as Hamming code, parity check code) to improve the error tolerance by increasing the amount of redundancy.

3.1.1. Optimize code logic
This type of research mainly adjusts and optimizes the encoding and decoding logic of parity and Hamming codes to improve error correction capability. Some researchers added one parity check code on the basis of Hamming code and ensured its different forms by modifying the same adjacent correction factors to achieve error location, which could realize detecting two and three adjacent errors [5]. [6] added the unit matrix optimized coding and decoding matrix on this basis. It could achieve two-bit error detection as well as multi-digit adjacent error detection while the number of codes remains unchanged. [7] used the measure of solving the Boolean-function satisfaction condition problem (SAT) to design the check matrix H, which can detect 2-3-bit errors and correcting one-bit errors when the amount of redundancy is the same as that of ordinary Hamming codes. In [8], the data to be protected were divided into information bits and key bits and decoded separately, while the decoding H matrix was optimized to merge the unused row data. Two-bit error correction can be realized with the same amount of redundancy as the Hamming code.

3.1.2. Increase the amount of redundancy
In addition to optimizing the codec logic, some studies have improved the error tolerance by increasing the amount of EDAC redundancy. For error detection capability, [9] proposed to group the parity check matrix, part of which was used to detect the type of error occurred (number of error bits, whether it is an adjacent error, etc.), and the other part was used to search the location of the error, causing up to 7 bits of error could be detected. [10] used OLS code to replace the parity check code, and also grouped the code into two parts for error type and error location separately. [11] proposed to divide the multi-bit data into several groups and group them for Hamming code coding, which increases the error correction and error detection capability of the same protected data by reducing the coding granularity, but causes an increase in the amount of redundancy. To detect more errors, [12] proposed the method of PBD (per bit detection), which interleaves a data into groups (i.e., non-adjacent data into one group) and assigns error detection codes to each group, improving the error detection efficiency. In order to improve the error correction capability, [13] proposed Matrix code, which treated the data to be protected as a matrix form, and used Hamming code to protect the row data and parity check code to protect the column data. It can realize the detection of three errors and correction of three errors. On this basis, CLC [14] and CLC-E [15] used parity check code plus Hamming code for the data in the same matrix form. Column data (including error correction code) were protected by parity code, which could detected more errors and correct up to 5-bit errors, and in addition, [15] proposed that an additional error correction process can be performed to achieve higher error correction capability. [16] adopted the same idea and proposes the PCoSA approach, which protected column data (including error correction code) also by Hamming code and parity, achieving 7-bit error correction but the area, power consumption, and delay overhead are large. Using the codec idea of 3 bit code, [17] continued the data protection form of Matrix code, which can achieve 6-bit or 12-bit error correction according to the form of data matrix.
Table 1 Comparison of metrics of some fault-tolerant methods (16bit)

| Method   | Reliability | Area ($\mu$m²) | Delay (ns) | Power consumption (mw) | Number of codes |
|----------|-------------|----------------|------------|------------------------|-----------------|
| RM       | Correction: 3 | 4816           | 2.86       | 0.774                  | 16              |
| Matrix   | Dection: 3, Correction: 3 | 1388           | 1.16       | 0.06                   | 16              |
| CLC      | Dection: 4, Correction: 5 | 1786           | 1.68       | 0.100                  | 24              |
| CLC-E    | Dection: 4, Correction: 7 | 3752           | 1.40       | 0.913                  | 24              |
| PCoSA    | Dection: n, Correction: 7 | 10579          | 2.34       | 0.891                  | 48              |
| PBD      | Dection: 7, Correction: 1 | 569            | 0.73       | 0.041                  | 4               |
| [17](2)  | Correction: 6 |                |            | 0.79                   | 24              |

Since these studies used different evaluation criteria, only some of the studies with the same evaluation criteria were selected for comparative analysis in this paper. These methods (for 16bit data protection) are compared here for reliability, area, latency, and power consumption in table 1. It can be seen that PCoSA performs better in terms of reliability, but has the largest area, power consumption and latency loss compared with other approaches; CLC-E adds two error corrections relative to CLC, which can correct more errors but consumes more power than CLC; and PBD can only perform one bit error correction, but has the smallest redundancy and the lowest area, power and latency metrics. This also shows that improving the error detection capability has less power consumption, latency penalty than improving the error correction capability.

3.2 Minimize latency, power consumption and area loss while ensuring fault tolerance reliability

Area, latency and power consumption are mainly influenced by the following factors:

- The coding redundancy in memory space. More redundancy will result in increased storage usage.
- The complex encoding and decoding logic. It is generally located on the critical path of the Load/Store instructions. This causes increased latency in instruction execution
- Encoding and decoding require sophisticated circuit support. Frequent coding and decoding operations will cause power cost.

Therefore, to resolve the above problems, a large amount of research has been conducted to achieve optimization of power consumption, area and latency loss mainly by optimizing the structure of Cache, such as simplifying the read/write path logic and storage strategy, while guaranteeing the fault tolerance reliability (codec logic).

To address the storage problem of EDAC, [18] proposed to separate ECC code from EDC code. EDC was stored in Cache for error detection, while ECC was stored in main memory for error correction when an error occurred. FIFO policy was applied to store ECC. Continuing the idea of separating ECC and EDC, [19] allocated storage blocks for ECC in Cache along with data, which could be stored according to coded data to determine the location and state, thus reducing the area overhead.

Besides, to optimize the latency problem caused by decoding on the critical path, [20][21] took advantage of the parallel reading of data in the Cache and decode before the Tag comparison selection data, which saved the waiting time for Tag comparison of data and also enhances the detecting probability of useless data. PSC-Cache [22] stepped further by placing both encoding and decoding logic in Tag comparison before selecting data, which could encode and decode multiple data simultaneously, multiplex the coding and decoding logic, and reduce the overhead. EUC [23]
combined the above two approaches, separating ECC and EDC, while increasing the coding granularity as well as the amount of read data decoding data, and classifying them into three coding modes according to the different granularity of coding: minimum area, minimum latency, and highest reliability. [24] added two registers to the original Cache structure, one was to store the iso-or of the data in the Cache and the other was to store the iso-or of the dirty data replaced out, optimizing the write read logic of the Cache. [25] proposed two methods of using parity, the first was to use parity only on the read path to improve reliability while reducing power consumption, and the second was to protect the smaller the granularity of data with higher the reliability.

Table 2 Summary of metrics of some fault-tolerant methods

| Method       | Configuration                                                                 | Power consumption (%) | Reliability (%) | Area (%) |
|--------------|-------------------------------------------------------------------------------|-----------------------|----------------|---------|
| SEA Cache    | L1Caches:Split I/D cache, 32 KB, 4-way, 32B line, L2Cache 1MB cache, 8-way, 64B line, | 19.4                  | 1              | 12.5    |
| REAP-cache   | L1Caches:Split I/D cache, 32 KB, 4-way, 64B line, L2Cache 1MB cache, 8-way, 64B line, | 2.7                   | 171×           | ≤1      |
| CLEAR        | L1Caches:Split I/D cache, 32 KB, 4-way, 64B line, L2Cache:256KB cache, 8-way, 64B line, | 1.52                  | 4.52 ×         | ≤2      |
| EUC-MinDelay | L1Caches:Split I/D cache, 32 KB, 4-way, 64B line, L2Cache:256KB cache, 8-way, 64B line, | 16.1                  | 1              | 8.1     |
| CPPC         | L1 D cache 32KB, 2-way, 32 byte lines, L1 I cache 16KB, 1-way, 32 byte lines, L2 cache 1MB unified, 4-way, 32 byte lines, | 7                      | 1              |         |

(*Compare to the baseline)

As can be seen from the table 2 above, these methods differ distinctly in power consumption, area and reliability due to the various configurations and EDAC codes used.

There are also several works that have been designed for other storage medium. For the STT-SRAM Cache, [26] used orthogonal Latin square code (OLSC) to encode the data, and LCLL and VnC are used for read and write strategies respectively, while the storage space of ECC was optimized to reduce the latency and area. For the storage on FPGAs, [27] used Column Mux to encode the data separately, while the correction factor was multiplexed to reduce the reliability problems caused by adjacent errors.

4. Fault-Tolerant Protection of Cache Tag

The data stored in Cache is divided into data and Tag, and most of the studies focus on the fault-tolerant design of data in Cache. Since an error in the Tag comparison process will lead to fault in reading or writing data, and even affect the data in main memory, which will seriously affect the
normal execution of instructions, several studies also focus on the fault-tolerant design of Tag. In addition, Tag is located in the critical read and write path of Cache, complex coding and decoding logic will bring delay problems, will affect the speed of the processor. Hence, Tag-based fault tolerance studies mainly start from simple EDAC codes or spatial redundancy.

Based on the idea of spatial redundancy, TRB [28] provided a small capacity buffer for the Cache Tag to provide a copy for the Tag of dirty rows, so as to prevent Tag comparison errors when the data was replaced. To extend the range of protectable Tag [29], the memory space in the cache-adjacent group was expanded to store Tag copies. In group-adjacent Cache, the above strategy requires index bits to locate Tag copy locations, so [30] applied this strategy in direct mapping Cache to reduce the storage Tag copy space. RAW-Tag [31] adopted TRB active replication to ensure that copies of dirty row Tag all existed and were stored in different ways of adjacent sets, using a complex replacement strategy to prevent simultaneous errors in the master copy. In addition to spatial redundancy, Tag could also be protected by simple EDAC. FastTag [32] chose to directly encode the Tag in the address and then compare it with the Tag in the Cache in terms of code distance, which reduced the overhead problem caused by repeated encoding and decoding, compared with the common one of decoding the EDAC-protected Cache Tag and then comparing it. Smartag [33], on the other hand, took advantage of the feature that the high bits of the address are basically the same, so the high bits of the Tag were designed to be shared by the ECC and the address, saving the storage space of the ECC.

Table 3 Comparison of metrics of some fault-tolerant methods in Tag

| Method   | Category               | Power consumption (%) | Correction capability | Area (%) |
|----------|------------------------|-----------------------|-----------------------|----------|
| TRB      | Spatial redundancy     | 19.9                  | 91.5                  | 16.3     |
| SimTag   | Spatial redundancy     | 11                    |                       |          |
| RAW-Tag  | Information Redundancy | 6.9                   | 97.9                  |          |
| FastTag  | Information Redundancy | 40.8**                |                       | 25.6**   |
| SMARTag  | Information Redundancy | 7                     | 99.6                  |          |

(*Compare to the baseline)
(\(^{b}\)Reduction compared to baseline)

From the relevant data in the table 3, it shows that the fault tolerance study for Cache is more concerned with the protection rate of Tag data.

5. Conclusion
As a key memory unit in processor, cache plays a decisive role in processor performance. It is located near the pipeline, and data can be easily and quickly forwarded to other units in case of errors. So the reliability design of Cache is critically required for processors running in radiation environment. The problem of increasing area, power and latency due to reliable design also needs to be considered as a compromise. Therefore, the current work focuses on either improving reliability or alleviating the area, power, and latency problems. In terms of reliability, it is now possible to achieve multi-bit correction like seven-bit fault tolerance; in terms of reducing the area, power and latency, there are various pragmatic methods such as separate EDC/ECC, the coding and decoding logic simplification, and the cache read/write path optimization. Typically, these approaches are applied to embedded processors or SoCs because they are more concerned with power consumption and real-time data. However, there are still some shortcomings in these researches, such as the inability to detect autonomously, the imbalances of the four indicators, etc.

As the semiconductor technology evolves, some new storage media emerge, such as STT-RAM type Cache, storage cells on FPGAs, etc. Future fault-tolerant technologies will probably be more flexible and adaptable to different storage media.
References

[1] R. W. Hamming, "Error detecting and error correcting codes," Bell Syst. Tech. J., vol. 26, no. 2, pp. 147-160, Apr. 1950.
[2] Tam S., Multiple Bit Error Correction[C]// IEEE Conference on Information & Communication Technologies. IEEE, 2004.
[3] S. Reed, "A class of multiple error correcting codes and decoding scheme" IRE Trans, vol. U-4, pp. 38-49, Sep 1954.
[4] DE Muller, "Application of Boolean algebra to switching circuit design and to error detection," IRE Trans, EC-3, pp. 6-12, Sep 1954.
[5] A Sánchez-Macián, Reviriego P., Maestro J A., Hamming SEC-DAED and Extended Hamming SEC-DED-TAED Codes Through Selective Shortening and Bit Placement[J]. IEEE Transactions on Device and Materials Reliability, 2012.
[6] Neale, A, Sachdev, et al. A New SEC-DED Error Correction Code Subclass for Adjacent MBU Tolerance in Embedded Memory[J]. Device & Materials Reliability IEEE Transactions on, 2013.
[7] Cui Y., Lou M., Xiao J., et al. Research and implementation of SEC-DED Hamming code algorithm[C]// Tencon IEEE Region 10 Conference. IEEE, 2014.
[8] Namba K., Lombardi F., A Single and Adjacent Error Correction Code for Fast Decoding of Critical Bits[J]. IEEE Transactions on Computers, 2018.
[9] Das A., Touba N A. Low Complexity Burst Error Correcting Codes to Correct MBUs in SRAMs[C]// The. 2018:219-224.
[10] Das A., Touba N A. Layered-ECC: A Class of Double Error Correcting Codes for High Density Memory Systems[C]// 2019 IEEE 37th VLSI Test Symposium (VTS). IEEE, 2019.
[11] Antony B. Modified Hamming Codes with Double Adjacent Error Correction along with Enhanced Adjacent Error Detection.
[12] Goerl R C., Villa P., Poehls L B., et al. An efficient EDAC approach for handling multiple bit upsets in memory array [J]. Microelectronics Reliability, 2018, 88.
[13] Argyrides, C., Pradhan, D. K., Kocak, T. Matrix Codes for Reliable and Cost Efficient Memory Chips[J]. IEEE Transactions on Very Large Scale Integration Systems, 2011, 19(3):420-428.
[14] Castro H., Silveira J., Coelho A., et al. A correction code for multiple cells upsets in memory devices for space applications [C]// New Circuits & Systems Conference. IEEE, 2016.
[15] Silveira F, Silveira J, et al.. An Extensible Code for Correcting Multiple Cell Upset in Memory Arrays[J]. Journal of Electronic Testing, 2018, 34:417-433.
[16] DF A., Dm A., Rg B., et al. PCoSA: A product error correction code for use in memory devices targeting space applications - ScienceDirect[J]. Integration, 2020, 74:71-80.
[17] Li J Q., Xiao L Y., Guo J., et al. Efficient Implementations of Multiple Bit Burst Error Correction for Memories [C]// 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT). IEEE, 2018.
[18] Yoon D H., Erez M. Flexible cache error protection using an ECC FIFO[C]// Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, November 14-20, ACM, 2009.
[19] Hong J., Kim S. Smart ECC Allocation Cache Utilizing Cache Data Space[J]. IEEE Transactions on Computers, 2017, 66(2):368-374.
[20] Cheshmikhani E., Farbeh H., Asadi H. Enhancing Reliability of STT-MRAM Caches by Eliminating Read Disturbance Accumulation[C]// 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). 2019.
[21] Farbeh H., Monazzah A. CLEAR: Cache Lines Error Accumulation Reduction by exploiting invisible accesses[J]. Microelectronics Journal, 2019, 90(AUG.):123-132.
[22] H Farbeh, Miremadi S G. PSP-Cache: A low-cost fault-tolerant cache memory architecture[C]// Design Automation and Test in Europe, 2014.
[23] Farbeh H., Delshadtehrani L., Kim H., et al. ECC-United Cache: Maximizing Efficiency of Error
Detection/Correction Codes in Associative Cache Memories[J]. IEEE Transactions on Computers, 2020, PP(99):1-1.

[24] Manoochehri M, Annavaram M, Dubois M. CPPC: Correctable parity protected cache[C]// International Symposium on Computer Architecture. IEEE Computer Society, 2011.

[25] Ko Y, Jeyapaul R, Kim Y, et al. Guidelines to design parity protected write-back L1 data cache. IEEE, 2015.

[26] Qureshi M A, J Park, Kim S. SALE: Smartly Allocating Low-Cost Many-Bit ECC for Mitigating Read and Write Errors in STT-RAM Caches[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2020, PP(99):1-14.

[27] Rahul K, Yachareni S. Low Power and Area efficient Forward Error correction code for SRAM in FPGAs [C]/ 2020 8th International Electrical Engineering Congress (iEECON). 2020.

[28] Hu J, Wang S. REPLICATING TAG ENTRIES FOR RELIABILITY ENHANCEMENT IN CACHE TAG ARRAYS.; US20140325138 [P]. 2014.

[29] Hong J, Kim J, Kim S. Exploiting Same Tag Bits to Improve the Reliability of the Cache Memories[J]. IEEE Transactions on Very Large Scale Integration Systems, 2015, 23(2):254-265.

[30] Archana M, Kashwan K R. 2016 International Conference on Recent Trends in Information Technology (ICRTIT) - An efficient architecture for improved r [J]. 2016:1-4.

[31] Farbeh H, Mozafari F, Zabihi M, et al. RAW-Tag: Replicating in Altered Cache Ways for Correcting Multiple-Bit Errors in Tag Array [J]. IEEE Transactions on Dependable and Secure Computing, 2017, PP(99):1-1.

[32] Reviriego P, Pontarelli S, Ottavi M, et al. FastTag: A Technique to Protect Cache Tags Against Soft Errors[J]. IEEE Transactions on Device & Materials Reliability, 2014, 14(3):935-937.

[33] Ghaemi S G, Ahmadvour I, Ardebili M, et al. SMARTag: Error Correction in Cache Tag Array by Exploiting Address Locality[C]/ 2018:1658-1663.