Self-Checking Cache Memory: Enhancing Reliability and Error Detection in digital Systems

Mohamed A. Abufalgha
Faculty of Engineering, Misurata University, Libya
m.a.abufalgha@eng.misuratau.edu.ly

Received: 15.10.2023 Published: 29.10.2023

Abstract:

Cache memory plays a crucial role in modern computing systems by providing fast access to frequently accessed data. However, the reliability of cache memory is of utmost importance to ensure data integrity and system stability. This paper presents a novel approach called "Self-Checking Cache Memory" that aims to enhance the reliability and error detection capabilities of caching systems. The proposed self-checking mechanism is based on the integration of error detection codes within the cache memory architecture. By incorporating an error detection code, the cache memory can identify and report errors that may occur during data retrieval or storage operations. Berger code are explored and evaluated for its effectiveness in detecting errors. Additionally, the paper introduces a parallel check code generation technique by partitioning the cache memory content into two segments: one for data and the other for check bits. It generates the complement of the check codes for the data in each reading task. This approach enables efficient error detection without compromising the performance of the cache memory system. The VHDL language has been employed to verify the self-checking scheme. The experimental results demonstrate that the self-checking cache memory design outperforms traditional cache memory systems in terms of error detection capability and reliability. The proposed design achieves shorter latency for error detection and requires less area compared to conventional error detection schemes; such as hardware redundancy and time redundancy.

Keywords: Self-checking scheme, soft errors, reliability, berger code, error detection code, cache memory.

الذاكرة المؤقتة ذات الفحص الذاتي: تعزيز الموثوقية وكشف الأخطاء في الأنظمة الرقمية

د. محمد أبوالقاسم أبوفلغة
كلية الهندسة - جامعة مصراتة

الملخص:

ذاكرة التخزين المؤقتة (كاش) تلعب دورًا حاسمًا في أنظمة الحوسبة الحديثة من خلال توفير وصول سريع إلى البيانات التي يتم التعامل معها بشكل متكرر. ومع ذلك، فإن موثوقية ذاكرة الكاش تعد أمرًا بالغ الأهمية لضمان سلامة البيانات واستقرار النظام. يقدم هذا البحث نهجًا جديدًا يسمى "ذاكرة الكاش ذات الفحص الذاتي"، والذي يهدف إلى تعزيز موثوقية وقدرات كشف الأخطاء العشوائية في أنظمة التخزين المؤقتة. تعتمد آلية الفحص الذاتي المقترحة على دمج شفرات كشف الأخطاء ضمن معمارية ذاكرة الكاش، من خلال إدراج شفرة لكشف الأخطاء، يمكن لذاكرة الكاش ذات التفحص الذاتي تحديد والإبلاغ عن الأخطاء التي قد تحدث أثناء عمليات استرجاع البيانات أو تخزينها. في هذا البحث تم اختيار شفرة بيرجر لفعاليتها في كشف الأخطاء. تتيح هذه الطريقة الكشف الفعّال عن الأخطاء دون المساس بأداء نظام ذاكرة الكاش. تم استخدام لغة VHDL للتحقق من نظام الفحص الذاتي لذاكرة الكاش. تظهر النتائج التجريبية أن تصميم ذاكرة الكاش ذات الفحص الذاتي يتفوّق على أنظمة ذاكرة الكاش التقليدية في قدرته على كشف الأخطاء وموثوقيته.

الكلمات المفتاحية: النظام ذاتي الفحص، الأخطاء الناعمة، الموثوقية، شفرة بيرجر، شفرة كشف الأخطاء، الذاكرة المؤقتة.

1. Introduction.

As technology scales down into the deep submicron regime, the susceptibility of digital circuits to soft errors increases. This phenomenon can be attributed to the decrease in transistor size, which reduces the amount of charge required to change the state of a transistor. Consequently, transistors become more vulnerable to interference from cosmic rays and radiation sources, leading to bit flips in stored data. Additionally, as transistor size decreases, the distance between adjacent transistors also shrinks, resulting in increased crosstalk and interference between neighboring circuits. These factors collectively contribute to the heightened susceptibility to soft errors in smaller transistors (Mano, 1993; Raji et al., 2016).

Given these new requirements, the ability of digital systems to self-check becomes highly desirable. Self-checking systems are automated processes designed to monitor and verify the accuracy and integrity of data, information, or operations. They find applications in various industries, such as healthcare,

finance, manufacturing, and transportation, where timely detection and correction of errors or discrepancies are essential. These systems employ algorithms, sensors, and other technologies to continuously monitor system inputs and outputs. They can generate alerts or notifications when errors are detected, allowing operators to take corrective action before serious consequences arise. In today's fast-paced and complex business environment, self-checking systems have become increasingly important for ensuring accuracy and reliability (Autran et al., 2012).

Soft errors in memories, also known as single-event upsets (SEUs), occur when the stored data in a memory cell is altered temporarily due to the impact of external radiation or electrical noise. Soft errors can affect various types of memories, including cache memories, dynamic random-access memory (DRAM), static random-access memory (SRAM), and non-volatile memory (NVM) technologies like flash memory. The susceptibility of a memory cell to soft errors depends on several factors, including its technology, size, voltage levels, and the surrounding environment. DRAM cells are particularly vulnerable to soft errors due to their small size and high integration density. SRAM cells, on the other hand, are less susceptible but can still experience soft errors. The susceptibility of cache memory to soft errors depends on several factors, including its size, technology, access patterns, and proximity to the processor. As cache memory is located on the chip alongside the processor, it is exposed to higher levels of radiation and electrical noise compared to other memory components. Additionally, the cache memory cells are often implemented using high-speed and low-power technologies, which can make them more sensitive to radiation-induced bit flips (Autran et al., 2012).

Self-checking technique enables online detection of faults during the normal operation of the system. It can be implemented using different methods, including information redundancy, hardware redundancy, and time redundancy. In this context, information redundancy involves adding extra bits to the data bits, protected by error-detecting codes, and continuously verifying their correctness using a checker. This approach enables the detection of errors as soon as they occur, preventing their propagation throughout the entire system. Additionally, error indicators can be utilized to trigger error recovery procedures (Marouf & Friedman, 1993).

Implementing error detection codes in self-checking cache memories typically involves utilizing algorithms such as parity codes, cyclic redundancy checks (CRC), or more advanced codes like Berger codes. The choice of the error detection code depends on factors such as the desired error detection capability, computational overhead, and implementation complexity (Lala, 2001).

In this paper, we aimed to leverage an existing technique and apply it to enhance and improve the reliability of a cache memory. This technique has been employed in various similar circuits, including microprogramming control units, Random Access Memories (RAMs), and register files.

2. Berger code.

The applicability of AUED codes to CED circuits would be significantly limited without the existence of a self-checking checker design for them. Presently, the Berger code is the most commonly utilized code in applications that require a systematic code. The Berger code (BC) is an optimal separable code that can detect all unidirectional errors. A Berger codeword with a length of n bits comprises I information bits and k check bits, where [k = log_2 (I+1)] and n = I+k. To construct a codeword, a binary number is generated based on the number of ones in the I information bits. The bit-by-bit complement of this binary number is then appended to the information bits as check bits (Lala, 2001).

For example, if I = 1100101, k = [log_2 (7+1)] =3 so the Berger code must have a length of 10 (7+3), k check bits are derived as follows:

Number of 1s in the information bits = 4

Binary equivalent of 4 =100

The bit-by-bit complement of 100 is 011, which are the k check bits. Thus,

n= 1100101 011

I k

The k check bits can be generated as the binary number representing the number of 0's in the I information bits. Therefore, the check bits for Berger codes can be generated using two different schemes. The scheme that utilizes the bit-by-bit complement of the binary number corresponding to the number of 1's in the information bits is known as the B1 encoding scheme. The other scheme, which uses the binary number corresponding to the number of 0's in the information bits as check bits, is identified as the B0 encoding scheme (Acharya & Rani, 2018).

Berger code is valuable for encoding information bits in digital systems due to the following reasons:

1. It is a separable code, eliminating the need for extra decoders to extract the information bits from the codeword when they are required for processing.
2. It detects all unidirectional errors, which are more likely to occur in digital systems.

3. It is an optimal code in terms of the number of check bits required for I information bits among all separable codes that detect unidirectional errors (Acharya & Rani, 2018).

Over the years, several techniques have been developed for designing self-checking checkers for Berger code. "Fig. 1" illustrates the main components of a self-checking system connected to a combinational circuit. C1 represents a circuit that generates the complements of the check bits for the information bits. The two-rail checker circuit compares the k check bits with the outputs of the C1 circuit. If there is no error, the outputs of the checker are complementary (i.e., 01 or 10); otherwise, they are 00 or 11 (Koren & Krishna, 2021).

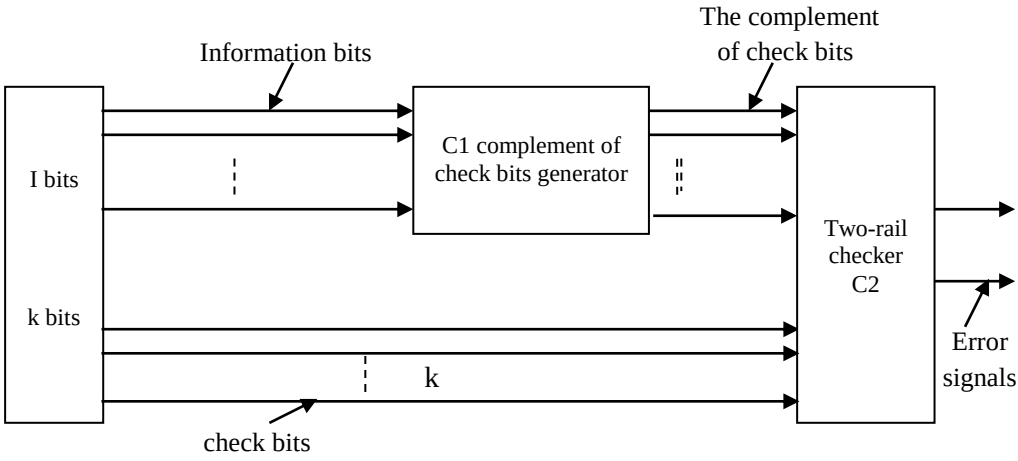


Figure (1): self-checking checker for Berger coed

3. Cache memory.

Cache memory is a high-speed memory type utilized to temporarily store frequently accessed data or instructions from a computer's main memory. It is positioned closer to the CPU than the main memory, enabling faster access times and improved system performance. Typically, computer systems have three levels of cache memory: L1, L2, and L3. L1 cache is the smallest and fastest, directly integrated into the CPU chip. L2 cache is larger but slower, located on a separate chip on the motherboard. L3 cache is even larger but slower, potentially shared between multiple cores or processors. The purpose of cache memory is to reduce the CPU's access time to data or instructions from the main memory. By storing frequently accessed data in cache memory, the CPU can retrieve it more quickly compared to accessing it from the main memory every time. This can result in significant performance improvements

for applications such as video games or graphics-intensive programs (Alsharef et al., 2021; Abayomi et al., 2020; Banday & Khan, 2014).

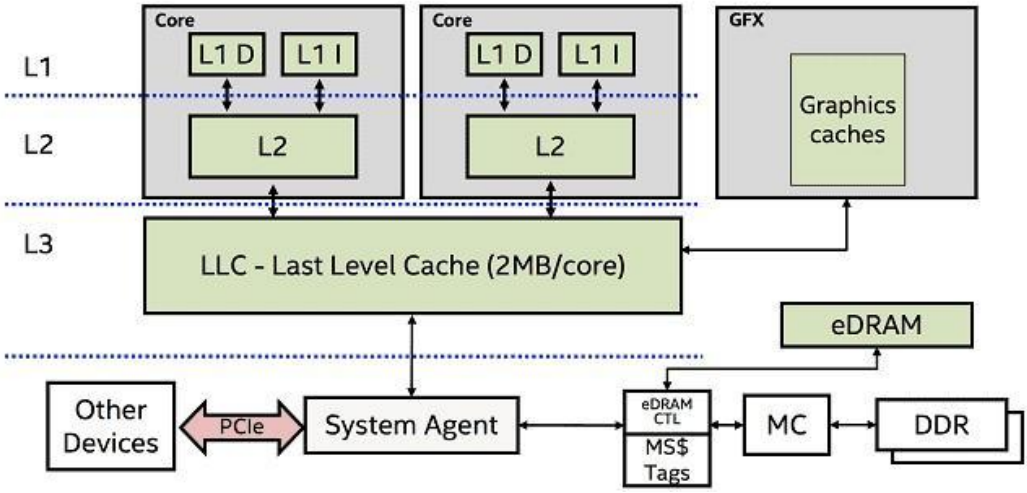
3.1 Cache memory levels.

There are several levels of cache memory (Stacpoole &Jamil, 2000; Alsharef et al., 2021):

1. Level 1 (L1) cache: L1 cache memory is built into the processor chip itself. It is the smallest and fastest type of cache memory, typically ranging from 8KB to 64KB in capacity. L1 cache memory aims to provide the processor with quick access to frequently used data and instructions, reducing the time it takes for the processor to retrieve information from the main memory. L1 cache memory operates at the same speed as the processor and is divided into two parts: instruction cache and data cache. The instruction cache stores frequently used instructions, while the data cache stores frequently accessed data. Both caches are accessed simultaneously by the processor, resulting in faster processing times.
2. Level 2 (L2) cache: This is a larger cache located on a separate chip near the CPU. Its size can range from a few hundred kilobytes to several megabytes, depending on the processor architecture. L2 cache memory operates at a slower speed than L1 cache but still significantly faster than the main memory. This provides a notable performance boost for applications requiring frequent access to data and instructions. In modern processors, L2 cache memory is often shared between multiple cores or threads, allowing them to access the same pool of cached data and instructions, further enhancing performance by reducing redundant caching.
3. Level 3 (L3) cache: L3 cache memory is situated between the CPU and the main memory (RAM) in a computer system. It is larger and slower than L1 and L2 caches, typically ranging from 4MB to 16MB in size. L3 cache memory is shared among all cores in a multi-core processor, enabling any core to access it when needed. L3 cache memory can significantly enhance the performance of applications requiring frequent access to large amounts of data, such as video editing, gaming, and scientific simulations. However, its effectiveness depends on factors such as the cache's size, speed, and integration with other system components.

Cache memory operates based on the principle of "locality of reference," which means that programs tend to access data and instructions that are close together in time and space. By storing frequently accessed data and instructions in cache, programs can run faster since they don't have to wait for data to be retrieved from the main memory. Overall, cache memory plays a crucial role in

enhancing computer performance by reducing the time required for the CPU to



access frequently used data and instructions (Chandra et al., 2019).

Figure (2): Cache memory levels L_1, L_2, and L_3

3.2 Soft Errors in Cache Memory.

Soft errors in cache memory occur when a bit in the cache memory is flipped due to external factors such as cosmic rays, alpha particles, or electromagnetic interference. These errors are temporary and do not cause permanent damage to the memory. Soft errors can occur in any type of cache memory, including L1, L2, and L3 caches. However, they are more likely to occur in larger caches due to their increased size and complexity. The impact of soft errors on cache memory depends on the severity of the error and the location of the affected bit. If a soft error occurs in a critical bit that stores important data or instructions, it can lead to system crashes or incorrect results. However, if the error occurs in a non-critical bit, it may go unnoticed or have minimal impact on system performance. To mitigate soft errors in cache memory, manufacturers use various techniques such as error-correcting codes (ECC), redundant arrays of independent memory (RAIM), and scrubbing. ECC adds extra bits to detect and correct errors while RAIM uses multiple redundant memories to store data. Scrubbing involves periodically checking and correcting errors in memory (Degalahal et al., 2005; Ko et al., 2017).

4. Self-checking cache memory.

Self-checking cache memory is a type of cache memory that incorporates built-in error detection and correction mechanisms. It is specifically designed to identify and rectify errors that may arise within the cache memory, eliminating the need for external error detection and correction techniques. Self-checking cache memory utilizes redundant data storage methods, such as parity checking or error-correcting codes (ECC), to store data within the cache. This enables the cache to detect errors in the stored data, such as bit flips or other forms of corruption. Upon detecting an error, the self-checking cache memory automatically corrects it using its integrated error correction mechanisms. This ensures that any errors within the cache are promptly addressed before they can disrupt the system's operation. Self-checking cache memory finds extensive usage in high-reliability systems, including aerospace and defense applications, where errors can lead to severe consequences. It is also employed in mission-critical applications, such as financial trading systems and medical equipment. Overall, self-checking cache memory provides an additional layer of protection against errors in the system, guaranteeing the accuracy and reliability of data stored within the cache (Taheri et al., 2022; Ostanin et al., 2015).

We applied the self-checking checker using Berger code on L3 cache memory because the probability of soft errors is more likely to happen in large circuits than smaller ones. In this paper our goal is to check every memory word whether it contains a soft error or not, before reaching the microprocessor. To achieve this goal, a special circuit is added to the cache memory to check if there is an error in the fetched block or not. The checking process can be done by checking the number of ones in the fetched memory word and compares it with the stored check bits (the check bits is constructed by appending the bit-by-bit one’s complement of the binary number corresponding to the number of ones in the memory word) we assumed that the data size is 15 bits, so the information bits and the check bits are 19 bits. Also, we used the B1 scheme, which means the check bits are the 1's complement of the number of ones in the information bits.

Figure 3 illustrates the structure of the self-checking cache memory, which comprises two essential circuits: The Complement of Check Bits Generating Circuit (CCBG) and the Two Rail Checker Circuit (TRC). The concept behind this design is that any memory word fetched undergoes two stages. First, it passes through the CCBG circuit, which extracts the number of ones (representing the one's complement of the check bits). Then, the stored check bits and their complements are forwarded to the TRC circuit, which compares them with the extracted check bits. The output of the TRC circuit serves as a control signal, determining whether the fetched memory word is permitted to

proceed to the microprocessor. This decision is based on the presence or absence of errors in the fetched memory word.

The CCBG circuit, depicted in "Fig. 4," generates the count of ones in a memory word. This circuit utilizes a group of full adders interconnected to tally the number of ones within the input information. In our scenario, the input information corresponds to the memory word fetched from the cache memory. Since our data consists of 15 bits, the resulting output will be 4 bits, representing the complement of the stored check bits. The output of this circuit serves as the input for the subsequent circuit, namely the TRC circuit.

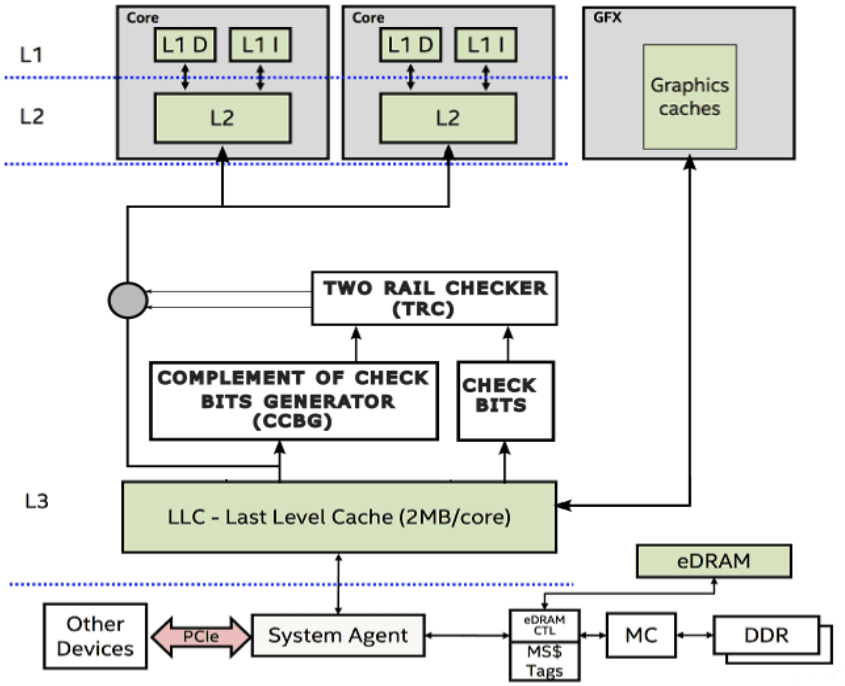


Figure (3): self-checking cache memory

The TRC circuit illustrated in Figure 5 is employed in digital electronic circuits to validate the accuracy of dual rail signals. In our case, the TRC circuit receives the check bits and their complements, comparing them bit by bit with the stored check bits (the one's complement of the count of ones) and the count of ones obtained from the CCBG circuit. If an error is present, the output of the TRC circuit will be (00 or 11). Conversely, if no error is detected, the output will be (01 or 10).

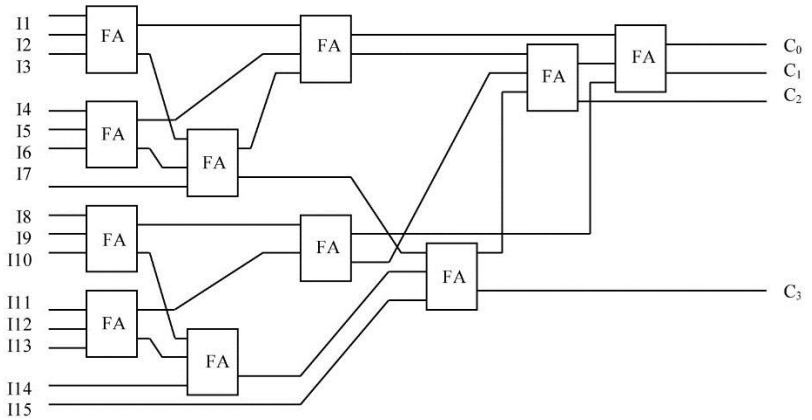


Figure (4): The complement of check bits generator

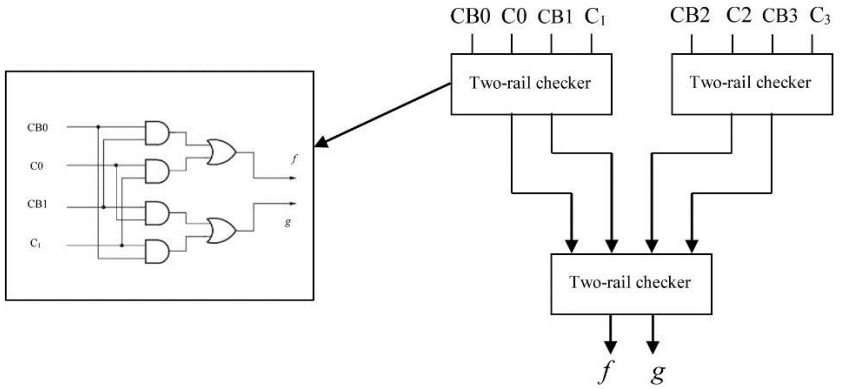


Figure (5): Two-rail checker circuit

5. The simulation results.

In this paper, we utilized the VHDL language to simulate the self-checking scheme added to the main circuit to enhance its reliability. VHDL, an acronym for Very High-Speed Integrated Circuit Hardware Description Language, is a standardized hardware description language widely employed in digital design and electronic systems. Initially developed by the U.S. Department of Defense in the 1980s, VHDL facilitates the design, simulation, and synthesis of intricate digital systems. VHDL serves as a formal and structured language for describing the behavior and structure of digital electronic systems. Designers

can specify the functionality, timing, and interconnections of digital components at various levels of abstraction, ranging from the system level down to the gate level. Consequently, VHDL is well-suited for describing a wide array of digital designs, including Application-Specific Integrated Circuits (ASICs), Field-Programmable Gate Arrays (FPGAs), and digital circuits. One of VHDL's key advantages lies in its ability to model and simulate complex digital systems before their physical implementation. Designers can employ VHDL to verify the correctness of their designs, conduct functional and timing simulations, and identify and resolve potential issues or bugs.

Figure 6 shows the contents of the cache memory (the information side) (I_0 *to* I_{14}) randomly stored to comprehend the functioning of the self-checking scheme. In addition to the stored information, check bits are also stored. Thus, in this case, we have 19-bit memory locations divided into two parts: the information part and the check bits part.

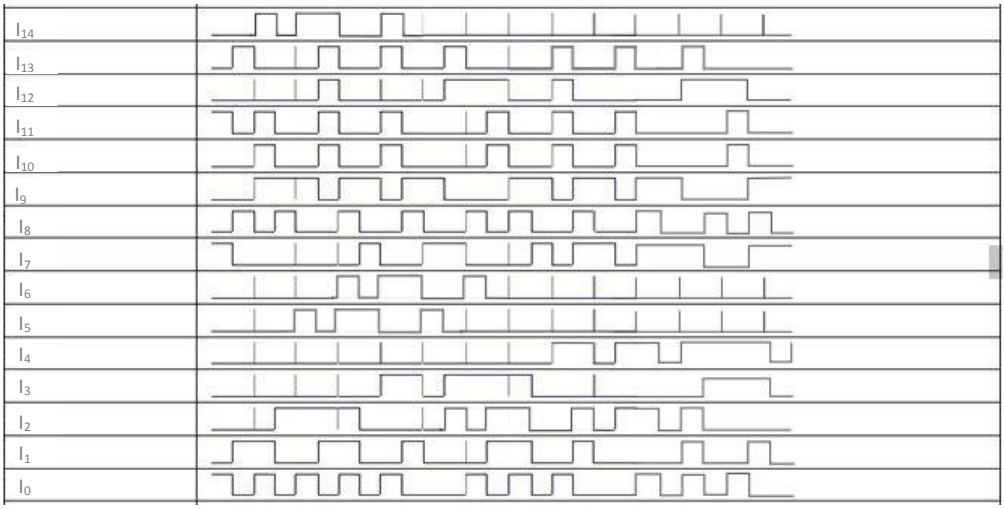


Figure (6): Memory words stored in the cache memory

Figure 7 illustrates the check bits part *(*$C.B_0$ *to* $C.B_3$*)*, where we implemented the one's complement of the number of ones in the memory word. We assume that the data has been read from the cache memory by the microprocessor, as depicted in Figure 8.

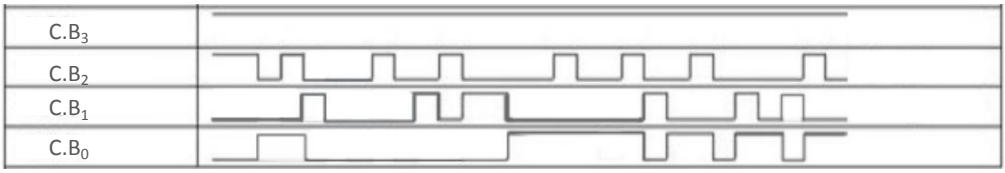


Figure (7): The check bits of the data that is stored in the cache memory

We intentionally inserted an error in the stored data to demonstrate that the scheme can detect flipped bits before they reach the microprocessor. So, a bit in the stored data has been changed from a 0 value into a 1 value. This caused a change in the number of ones in the data. CCBG circuit is responsible in calculating the number of ones in the data, and the output of this circuit has been shown Figure (9). These bits represent the complement of the check bits. So, this change in one bit will be discovered by comparing these bits with the check bits. The input of CCBG circuit is a copy of the fetched memory word that is intended to be sent to the microprocessor. However, with the presence of the self-checking scheme, sending this memory word is contingent upon the signal from the two-rail checker circuit (TRC) shown in Figure 5.

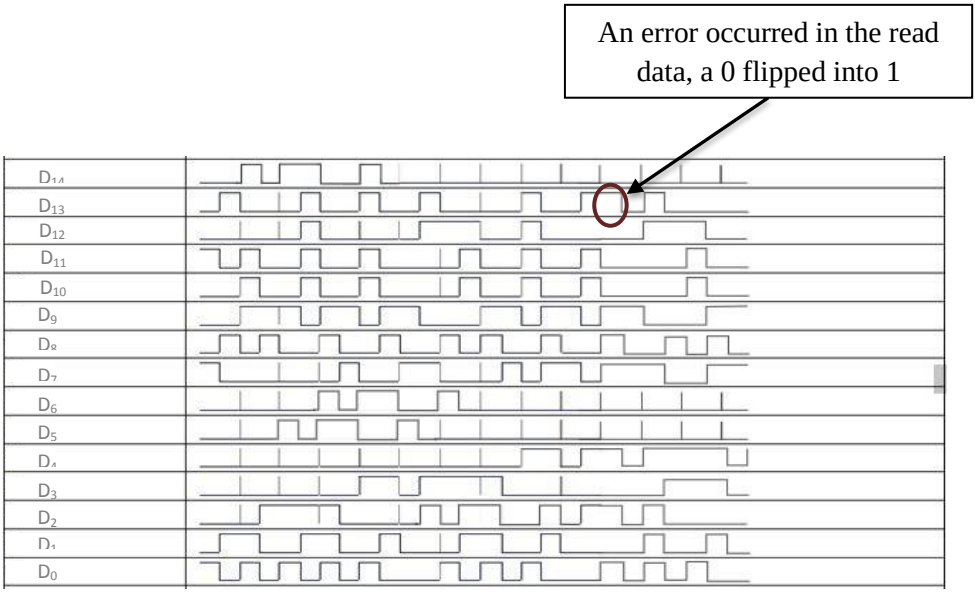


Figure (8): The data that fetched from the cache memory

Figure 10 demonstrates the process of error detection using the check bits: CB3, CB2, CB1, CB0 = 1010, which represents the one's complement of the number of ones in the information bits. Therefore, the expected count of ones should be 0101. However, the output of the CCBG circuit (the circuit that generating the complement of the number of the check bits) is shown in the same figure as C3 C2 C1 C0 = 0111, deviates from the expected value due to the inserted error in the data in Figure 8. As a result, due to the mismatching between the check bits and the output of the CCBG circuit, both f and g are equal to 0 (f and g = 00), indicating that there is an error has been occurred in the data, (a presence of an error in the data), so this memory word is discarded and cannot be sent to the microprocessor with presence of TRC circuit.

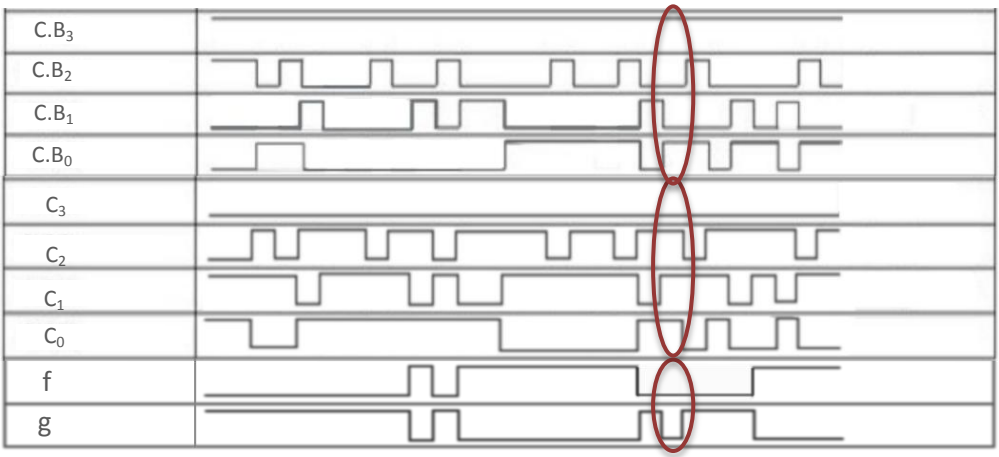


Figure (10): The number of ones in the data that fetched from the cache memory

The results showed that utilizing the Berger code for detecting unidirectional soft errors is an effective method to protect the circuit under test from this type of error. The challenge with detecting soft errors is that they do not occur consistently or at predictable intervals; instead, they appear randomly. Therefore, they are not easily detected through ordinary or offline testing methods. In this paper, an error was intentionally inserted into the data, and the suggested scheme successfully detected this error during normal operation. Additionally, the results indicated that all fetched data is tested before reaching the microprocessor, enhancing the reliability of the system by ensuring that no erroneous data is processed.

6. The conclusion.

In this paper, we utilized the information redundancy technique to construct a self-checking cache memory. Information redundancy is considered the most favorable type among error detection and fault tolerance techniques. It offers advantages such as lower area overhead compared to hardware redundancy (such as duplex or triple control units connected in parallel) and reduced time requirements compared to time redundancy. The Berger code was used in combination with information redundancy because it is an optimal code for handling all unidirectional errors.

The L3 cache memory type was chosen as the circuit under test due to its susceptibility to soft errors caused by its large size. The self-checking scheme was designed to be comprehensible and applicable to various storage devices. The results demonstrated that any changes occurring in the stored data, such as switching from 0 to 1 or vice versa, are captured by this scheme, preventing them from reaching the microprocessor.

VHDL language was employed to simulate the self-checking cache memory, as it is effective for this type of simulation, especially at the gate level, which is the case in this paper. The verification process using the self-checking scheme was successfully conducted by intentionally inserting an error in the fetched data to assess whether the error could be detected. Our findings revealed that the error was detected by the TRC circuit, which then sent a command signal to another circuit to discard the erroneous memory word.

The main advantages of this scheme are its simplicity, effectiveness, and minimal overhead in terms of area and time. However, despite the effectiveness of this method, there are some limitations that pose challenges when attempting to recover the original data without resorting to additional error correction techniques. Another limitation of the Berger code is its susceptibility to certain types of errors. Although it can effectively detect single-bit errors, it may fail to detect more complex errors, such as burst errors or multiple-bit errors that occur in close proximity. This limitation can compromise the overall reliability of the system, as undetected errors can propagate and lead to data corruption or system failures.

This work may extend to studying the trade-off between reliability and energy consumption resulting from the utilization of a self-checking scheme. Additionally, it could explore the area overhead and delay that are incurred by implementing this scheme.

References

Abayomi, Ademodi Oluwatosin, Olukayode, Ajayi Abayomi and Olakunle, Green Oluwole. (2020). An Overview of Cache Memory in Memory Management, Automation and Control and Intelligent Systems, Volume 8, Issue 3, doi: 10.11648/j.acis.20200803.11.

Acharya, Prasad & Rani, M. (2018). Berger code based concurrent online self-testing of embedded processors, Journal of Semiconductors. Vol. 39, doi: 115001. 10.1088/1674-4926/39/11/115001.

Alsharef, A., Garg, Sonia, Zahra, Syed, Jain, Pankaj, Arora, Monika and Gupta, Gaurav. (2021). Cache Memory: An Analysis on Performance Issues. 8th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, pp. 184-188.

Autran, Jean-Luc & Semikh, Sergey & Munteanu, Daniela & Serre, S. & Gasiot, Gilles & Roche. (2012). *Soft-Error Rate of Advanced SRAM Memories: Modeling and Monte Carlo Simulation*. IntechOpen, 10.5772/50111.

Banday, M. Tariq & Khan, Munis. (2014). A study of recent advances in cache memories. Proceedings of International Conference on Contemporary Computing and Informatics, IC3I. 398-403, doi: 10.1109/IC3I.2014.7019786.

Chandra, Y Poorna and S. Afreed. (2019). Literature survey on cache memory, International Journal of Advance Research and Innovative Ideas in Education, PP. 142-147.

Degalahal, V., Li, Lin, Narayanan,V., Kandemir, M. & Irwin, M. J., (Oct. 2005) Soft errors issues in low-power caches, IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 13, no. 10, pp. 1157-1166, doi: 10.1109/TVLSI.2005.859474.

Ko, Yohan & Jeyapaul, Reiley & Kim, Youngbin & Lee, Kyoungwoo, and Shrivastava. (2017). Protecting Caches from Soft Errors: A Microarchitect's Perspective. ACM Trans. Embed. Comput. Syst. 16, 4, Article 93, 28 pages. https://doi.org/10.1145/3063180

Koren, Israel & Krishna, C. Mani. (2021). *Fault-Tolerant Systems*, Second Edition, Morgan Kaufmann.

Lala, K. Parag. (2001). *Self-Checking and fault tolerant Digital Design*, First edition, Morgan Kaufman publishers,

Mano, M. Morris. (1993). *Computer System Architecture*, third edition, Prentice-Hall.

Marouf, M. A. and Friedman, A. D. (1993). Design of Self-Checking checker for Berger codes, IEEE transactions on computers, Volume 42, Issue 8.

Ostanin, S. & Kirienko, I. and Lavrov, V. (2015). A fault-tolerant combinational circuit design, IEEE East-West Design & Test Symposium (EWDTS), Batumi, Georgia, pp. 1-4, doi: 10.1109/EWDTS.2015.7493130.

Raji, M., Sabet, M. A. and Ghavami, B. (2019). Soft Error Reliability Improvement of Digital Circuits by Exploiting a Fast Gate Sizing Scheme, in IEEE Access, vol. 7, pp. 66485-66495, doi: 10.1109/ACCESS.2019.2902505.

Stacpoole, R. & Jamil, T. (May 2000). Cache memories, IEEE Potentials, vol. 19, no. 2, pp. 24-29, doi: 10.1109/45.839642.

Taheri, M. & Sheikhpour, S. & Mahani, A. and Jenihhin, M. (2022). A Novel Fault-Tolerant Logic Style with Self-Checking Capability, IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS), Torino, Italy, pp. 1-6, doi: 10.1109/IOLTS56730.2022.9897818.