ISSN: 0974-5823 Vol. 6 No. 1 January-June, 2021

International Journal of Mechanical Engineering

CODES FOR AUTOMATIC ERROR CORRECTION IN DATA ENTRY AND MATCHING HAVE BEEN IMPLEMENTED

[1]**Dr.Shaik Saidhbi,** Associate Professor, Department of computer science, Samara University, sfajju.syed@su.edu.et, Ehiopia

[2]**Adem ali Kabo,** Department of computer science, Samara University, ademali@su.edu.et, Ehiopia

[3]**Ahmed Adem ,** Computer Science, Samara University, aniadem.adem0@gmail.com,Ethiopia

Abstract

A computer is a system that compares new information with existing data in order to get the best possible result. Cache tag array lookup matching and translation lookaside buffer are two such examples. To simplify and speed up the process of matching data encrypted using error-correcting codes, we suggest a novel architecture in this work (ECC). The raw data and parity information that make up the codeword of an ECC created via encoding are expressed in a standard way. The suggested architecture uses parallel processing to compare data and parity in real time. In order to efficiently compute the Hamming distance, we introduce an unique butterfly-shaped weight accumulator (BWA), which significantly decreases both latency and complexity. To ensure that new information is consistent with existing information, the suggested architecture compares the two.

Keywords: "Data comparison, error-correcting codes (ECCs), Hamming distance, systematic codes, tag matching."

1. INTRODUCTION

Many computer operations rely on comparing data, including tag matching in cache memory and virtual-to-physical address translation in a translation lookaside buffer (TLB). Given its widespread usage, it is crucial to find a way to construct the comparison circuit with minimal hardware complexity. It is also common for the data comparison to be on the critical path since the outputs of components like caches and TLBs affect the order in which following operations in a pipeline are executed, which is intended to optimise system performance. Therefore, it is essential that the circuit be built to have minimal latency; otherwise, the components cannot be used as accelerators, and the system's performance suffers. Error-correcting codes (ECCs) are used by modern computers to secure data and increase reliability, but the sophisticated decoding operation that must precede the data comparison adds time and complexity to the critical route. That makes it that much more challenging to adhere to the aforementioned design requirements. However, the works that address this issue are not well-known in the literature since it has traditionally been dealt with inside industry for their goods, despite the fact that they are clearly necessary. However, in recent years, has drawn much greater attention from the scholarly community.

Encoding incoming data and then comparing it to returning encoded data is the most modern method for resolving the matching problem. As a result, the technique takes difficult decoding off the route to failure. The method does not check whether the information being obtained is an exact match for the information being input. Instead, it checks to see whether the error correctable range of the codeword that maps to the incoming information overlaps with the information that was returned. Since the number of unique bits separating the two codewords is an essential part of the Hamming distance computation, the saturate adder (SA) was developed to alleviate the need for an additional circuit during the testing process.

An important factor that might greatly improve efficiency was overlooked in this research, however: the fact that a genuine ECC codeword is often stored in a methodical fashion in which the data and parity parts are

fully isolated from one another. In addition, the SA adds to the overall circuit complexity by requiring that its output never exceed the number of visible faults by more than one.

In this condensed paper, we update the SA-based direct comparison design to address these issues and lower latency and device complexity. Specifically, we offer a low-complexity processing unit that quickly computes the Hamming distance, taking into account the features of systematic codes as we construct the resulting architecture. As a result, both the latency and hardware complexity are reduced, and this is true even when compared to the SA-based design.

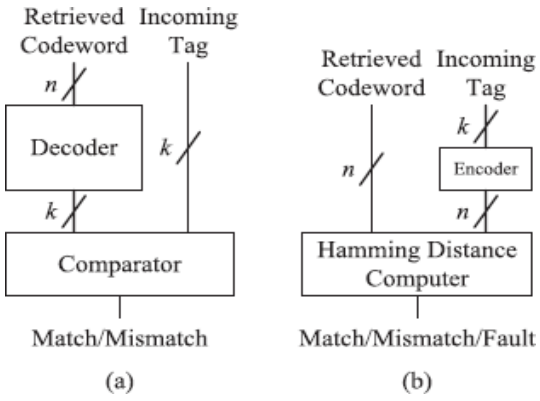
This summary will continue in the following format. In the second part, we examine the relevant literature. Section 3 outlines the planned architecture, whereas Section 4 presents the findings. Section 5 provides some last thoughts.

2. PREVIOUS WORKS

Both the conventional decode-and-compare structure and the more modern encode-and-compare structure based on direct comparison are described. For simplicity's sake, this article just covers cache memory tag matching, however the suggested architecture can easily extended to cover other related uses.

A. Structure for Decoding and Comparing Data

Let's pretend we have a cache memory where a k-bit tag is encoded by a (n, k) code and then stored as an n-bit codeword. The initial step in the decode-and-compare architecture depicted in Fig. 1 is decoding the n-bit returned codeword to retrieve the original k-bit tag (a). Having gotten the k-bit tag, it is then compared to the k-bit tag field of an incoming address to see whether a match exists. Since the recovered codeword needs to be processed by the decoder before being compared with the incoming tag, the crucial path is too time-consuming to be employed in a practical cache system designed for quick access. In addition, the complexity overhead is not inconsequential since the decoder is one of the most complex processing units.



"Fig. 1. (a) Decode-and-compare architecture and (b) encode-and-compare architecture."

B. System Design Based on Encoding and Comparing

It's important to keep in mind that decoding, which involves a number of steps like error identification or syndrome computation and error repair, is often more difficult and time-consuming than encoding. There is evidence from the implemented changes to back up the assertion. Encoding an incoming tag instead of decoding a recovered codeword is one way to get around problems with the decode-and-compare method. As shown in Fig. 1, an entering k-bit tag is first encoded to the appropriate n-bit codeword X, and then X is compared with a returning Y. (b).

The goal is not to verify if the two codewords are identical, but rather to count the number of differences between them. The Hamming distance d between the two codewords is then used to categorise the occurrences. We will refer to the maximum number of errors that can be fixed as tmax and the maximum

number of errors that can be found as rmax. The cases are briefly summarised here.

1) If d = 0, X matches Y exactly.

2) If 0 < d ≤ tmax, X will match Y provided at most tmax errors in Y are corrected.

3) If tmax < d ≤ rmax, Y has detectable but uncorrectable errors. It's possible that the cache would then report a system error to the CPU, prompting it to take the necessary measures.

4) If rmax < d, X does not match Y .

Given that computing the Hamming distance is an integral part of the aforementioned strategy, we have shown a circuit specifically designed to do so. To calculate the bitwise difference between X and Y, the circuit first performs XOR operations on every pair of bits in X and Y, as shown in Fig. 2. Here are several half adders (HAs) that count how many times two adjacent vector bits are both 1. By iteratively traversing the subsequent SA tree, we may calculate the total number of 1s. If the total value z in the SA tree rises over rmax, it is capped at rmax + 1. Specifically, the expression for z is as follows, where x and y are inputs.

The scope of d is represented by the sum total. The complexity of a SA is more than that of a regular adder due to the need of extra logic circuitry to handle the saturation requirements.

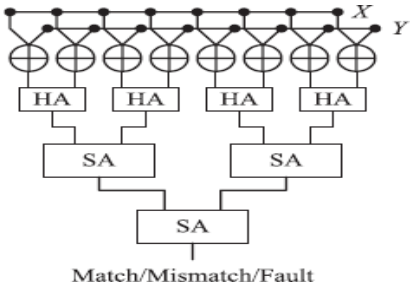


Fig. 2. Structure based on SA that allows for direct comparison

3. PROPOSED ARCHITECTURE

Using the properties of systematic coding, a novel architecture is shown in this part that may speed up and simplify data comparison. A new processing component is also included to further minimise latency and complexity.

A. Block Diagram

The rationale for the proposed layout is shown in Fig. 3. In order to encode the incoming data, the parity bits are added to the data stream.

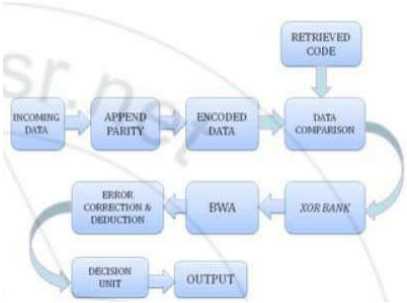
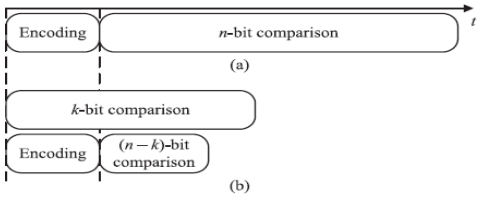


Fig3:Block Diagram

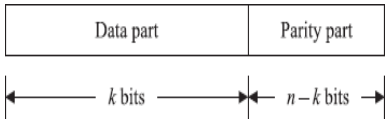
The returned information is then compared to the encoded information in memory. The XOR bank and Butterfly-shaped weighted accumulator are used to accurately count the number of bit transitions and calculate the total number of ones, which are then sent into the error correction and error deduction unit. Therefore, the source of the output is the decision unit.

B. Datapath Design

After the input tag has been encoded, the SA-based architecture checks for a match between two codewords. This indicates that the critical path involves a series of encoding and n-bit comparisons, as seen in Fig. 4. (a). Unfortunately, this ignores the fact that the ECC codeword really has the systematic structure seen in Fig. 5. As soon as the encoding is complete, the data part of a systematic codeword may be compared with the incoming tag field, but the parity part cannot be accessed until then. Knowing this, we may start comparing the k-bit tags before completing the remaining (n-k)-bits of parity. Figure 4 shows how the proposed architecture reduces overall latency by performing the encoding operation to generate the parity bits from the incoming tag at the same time as the tag comparison (b).



"Fig.4. Timing diagram of the tag match in (a) direct compare method (b) proposed architecture."



"Fig. 5. ECC codeword represented in a systematic manner."

C. The Hamming Distance and Its Computational Architecture

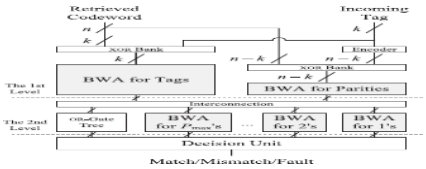
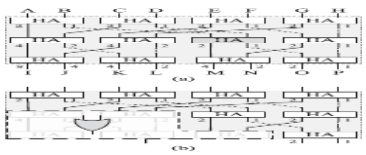


Fig. 6. The proposed architecture is codeword-optimized

Figure 6 depicts the suggested architecture based on the datapath design. It includes a number of butterfly-shaped weight accumulators (BWAs) that have been suggested to reduce the delay and complexity of the Hamming distance calculation. The BWA's primary job is to determine how many ones are present in the input stream. As illustrated in Fig. 7(a), it is made up of a cascade of HAs, with a weight assigned to each bit of a HA's output at each level.



"Fig. 7. Proposed BWA. (a) General structure and (b) new structure revised for the matching of ECC-protected data. Note that sum-bit lines are dotted for visibility."

Individually aggregating the carry bits and the sum bits from the prior stage requires the HAs of that stage to be connected in a butterfly arrangement. The inputs to a HA are usually the carry bits or sum bits determined in the preceding stage, with the exception of the first stage. When the HA's output bit is set, the route weight is proportional to the number of ones in the route leading to the HA. In Fig. 7(a), for instance, the number of 1's among the corresponding input bits (A, B, C, and D) is 2 if the carry bit of the gray-colored HA is set. In Fig. 6(a), step d yields the fraction of input bits that are ones:

$$d = 8I + 4(J + K + M) + 2(L + N + O) + P.$$

To simplify the circuit, we are not concerned with the precise Hamming distance, but rather the range to which it belongs. For instance, when rmax = 1, the same situation applies whether there are two or more than two 1s among the input bits. Then, as illustrated in Fig. 7, we may use a simple OR-gate tree to implement the functions of several HAs (b). The SA that uses forced saturation is at a disadvantage here.

As can be seen in Fig. 7, the carry-bit lines and the sum-bit lines do not intersect with one another. Modern technology offers several routing levels such that, regardless of how many bits a BWA uses, any overlaps may be easily resolved between carry-bit lines and sum-bit lines.

Here, we'll go into further depth on the architecture as a whole. When determining the Hamming distance, Fig. 6 shows how counting the number of ones in the bitwise difference vector between the data bits and the parity bits after each XOR operation is performed. Updated versions of the BWAs at the first level are shown in Fig. 7(b). These BWAs now provide an output from the OR-gate tree and multiple weight bits from the HA trees. These results are then sent to the network's second-level nodes for analysis. The result of an OR-gate tree is fed into the input of another OR-gate tree, with any remaining weight bits being fed into the BWAs of the second level in an inverse-weighted method. In the second level, the weight associated with each BWA is a power of two that is less than or equal to Pmax, where Pmax is the greatest power of two that is not more than rmax + 1. We may safely disregard the powers of two that are more than Pmax since the new BWAs OR together all of the weight bits associated with the fourth range.

D. Common Phrases for Describing Complexity

The complexity and latency of combinational circuits are very technique dependent. In addition, it is difficult to develop an analytical and entirely deterministic equation that depicts the relationship between the number of gates and the latency for both the suggested design and the standard SA-based architecture due to the inherent conflict between complexity and latency. Avoiding the difficulty of analytical derivation, we provide an equation that may be used to estimate complexity and latency by substituting in a few variables for the nondeterministic parts. To calculate the complexity (C) of the suggested design, one may use the following formula:

$$\begin{aligned} C &= C_{XOR} + C_{ENC} + C_{BWA}(k) + C_{BWA}(n-k) + C_{2nd} + C_{DU} \\ &\leq n + C_{ENC} + 2C_{BWA}(n) + C_{DU} \end{aligned}$$

Complexity of XOR bank, encoder, second-level circuits, decision unit, and BWA for n inputs are denoted by CXOR, CENC, C2nd, CDU, and CBWA(n), respectively. In particular, we can determine CBWA(n) using the recurrence relation, which looks like $C_{BWA}(n) = C_{BWA}(\lfloor n/2 \rfloor) + C_{BWA}(\lceil n/2 \rceil) + 2\lfloor n/2 \rfloor$

if we use CBWA(1) as our seed, and set it to 0. CBWA(a) + CBWA(b) CBWA(c) holds for a+b=c for any three positive integers a, b, and c. Due to the inequality and the fact that an OR-gate tree for n inputs is always simpler than a BWA for n inputs, CBWA limits both CBWA(k) + CBWA(n -k) and C2nd (n).

E. Latency Expressions in General

Specifically, we may write out an expression for L, the latency of the suggested architecture, as

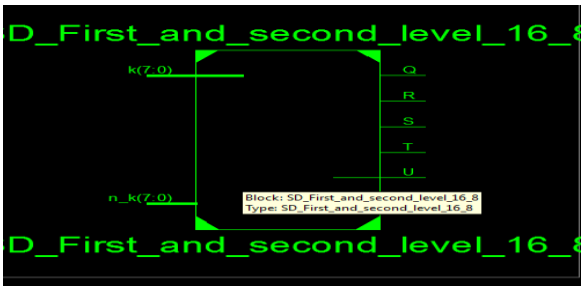
$$L \le \max\left[L_{XOR} + L_{BWA}(k), L_{ENC} + L_{XOR} + L_{BWA}(n-k)\right] + L_{2nd} + L_{DU}$$
$$\le \max\left(1 + \lceil \log_2 k \rceil, L_{ENC} + 1 + \lceil \log_2 (n-k) \rceil\right) + \lceil \log_2 n \rceil + L_{DU}$$

where LXOR and LENC are the latencies of the XOR bank and encoder, L2nd and LDU are the latencies of the second level circuits, and LBWA(n) is the latency of the BWA for n inputs. Keep in mind that the OR-gate tree and BWAs have latency bounds of [log2 n] for x n inputs at the second level. Some parts of the second level could begin early if one of the first level BWAs completes its work sooner. In a similar vein, the second-level OR-gate tree or certain BWAs may rush to provide their output to the decision unit so that it may start processing data immediately rather than waiting. This is because the critical path of the previous circuits might obscure some of L2nd and LDU, making L shorter than the stated expression.

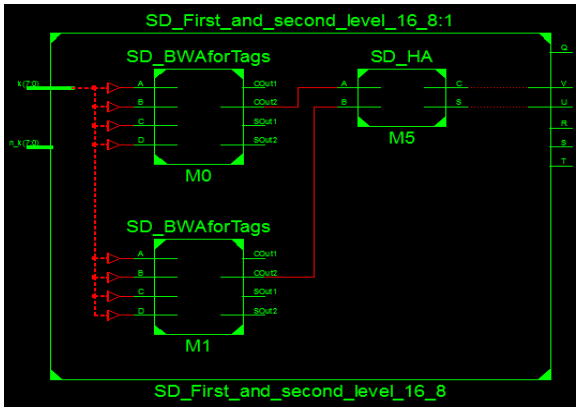
1. RESULTS

Taking into account realistic considerations, the suggested design successfully decreases both latency and hardware complexity. It is important to keep in mind that as codeword size rises, the suggested architecture's advantage over the SA-based one in reducing latency grows. The following explanation explains why: When comparing the proposed architecture to the SA-based one, it is clear that the latencies are dominated by HAs. With each doubling of the bit width, an additional SA or HA stage is required. Since a HA's critical path consists of a single gate rather than a SA's, the proposed design offers reduced latency than the SA-based alternative, especially for long code words.

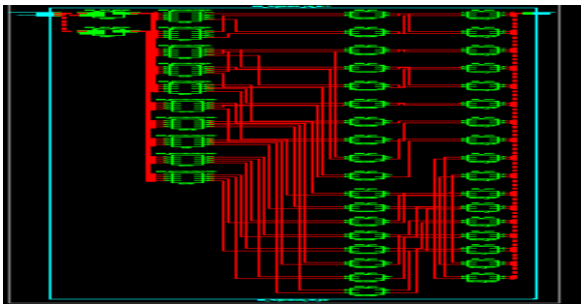
simulation results



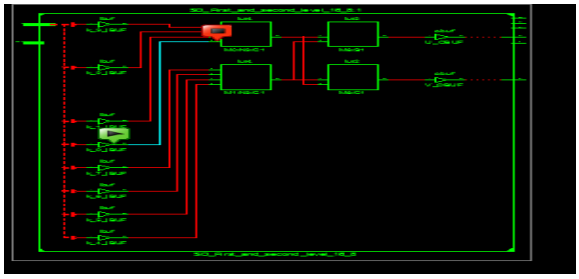
"fig 8: RTL Schematic"



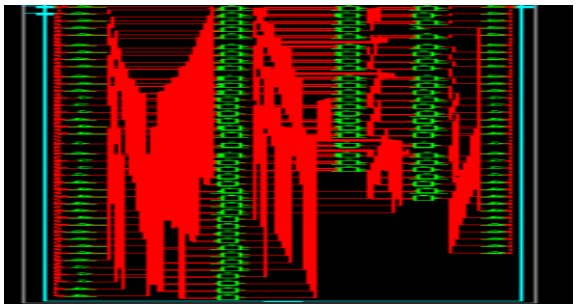
"Fig 9: Internal structure for 16_8 RTL schematic"



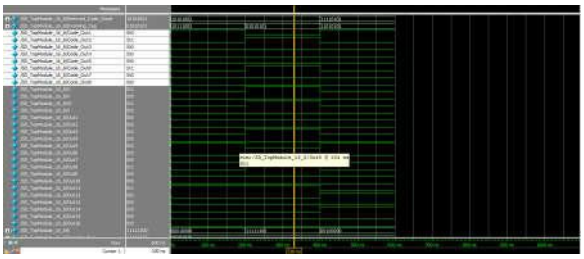
“Fig 10: Technology Schematic of 16_8”



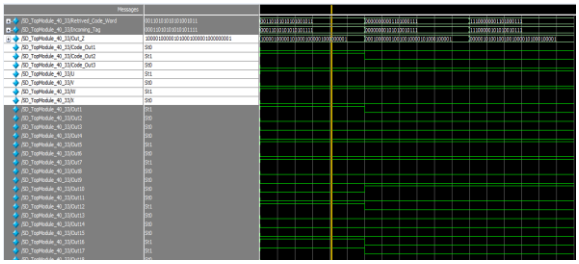
“Fig 11: Internal structure for 40_33 RTL schematic”



“Fig 12: Technology Schematic of 40_33 bits”



“Fig 13: Simulation result of 16_8 bits”



“Fig 14 Simulation result of 40_33 bits”

2. CONCLUSION

A novel architecture for comparing ECC-protected data has been revealed, which promises to cut down on device complexity and latency. The proposed architecture checks whether or not the received data coincides with the saved data and whether or not a certain number of mistakes have been repaired. In order to reduce waiting time, we parallelize the data comparison with the encoding process that generates the parity data. The systematic codeword's data and parity fields allow for parallel processing. To further reduce delay and complexity, a streamlined processing architecture has also been provided. The suggested architecture shows promise as a solution for comparing ECC-protected data due to its ability to significantly reduce latency and complexity. The suggested approach is not limited to the tag match of a cache memory, as shown in this short, but may be used in a wide variety of contexts where such a comparison is necessary.

REFERENCES

[1] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S.-L. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu, and D. Srivastava, “The 65-nm 16-MB shared on-die L3 cache for the dual-core Intel xeon processor 7100 series,” IEEE J. Solid-State Circuits, vol. 42, no. 4, pp. 846–852, Apr. 2007.

[2] J. D. Warnock, Y.-H. Chan, S. M. Carey, H. Wen, P. J. Meaney, G. Gerwig, H. H. Smith, Y. H. Chan, J. Davis, P. Bunce, A. Pelella, D. Rodko, P. Patel, T. Strach, D. Malone, F. Malgioglio, J. Neves, D. L. Rude, and W. V. Huott “Circuit and physical design implementation of themicroprocessor chip for the zEnterprise system,” IEEE J. Solid-State Circuits, vol. 47, no. 1, pp. 151–163, Jan. 2012.

[3] M. Tremblay and S. Chaudhry, “A third-generation 65nm 16-core 32-thread plus 32-scout-thread CMT SPARC processor,” in ISSCC. Dig.Tech. Papers, Feb. 2008, pp. 82–83.

[4] Y. Lee, H. Yoo, and I.-C. Park, “6.4Gb/s multi-threaded BCH encoder and decoder for multi-channel SSD controllers,” in ISSCC Dig. Tech.Papers, 2012, pp. 426–427. S. Lin and D. J. Costello, Error Control Coding: Fundamentals and Applications, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall,2004

[5] S. Lin and D. J. Costello, Error Control Coding: Fundamentals and Applications, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall,2004.

[6] G. Olocco and J. Tillich, "A family of self-dual codes which behave in many respects like random linear codes of rate 1/2," in Proceedings of the IEEE ISIT , 24-29 June 2001, p. 15.