Hard Systematic Error Correcting codes based Matching of Data Encoded Architecture with Low-Complexity, Low-Latency with FPGA Implementation

B.Varun[1], Ch. Chandra Rao[2], B.J Ranjitha[3], S.Neelima[4]

[1] Student, ECE Department, Gandhiji Institute of Science and technology, Andhra Pradesh, India
[2] Student, ECE Department, Gandhiji Institute of Science and technology, Andhra Pradesh, India
[3] Student, ECE Department, Gandhiji Institute of Science and technology, Andhra Pradesh, India
[4] Associate Professor, ECE Department, Gandhiji Institute of Science and technology, Andhra Pradesh, India

ABSTRACT

A new architecture for matching the data protected with an error-correcting code (ECC) is presented in this brief to reduce latency and complexity. Based on the fact that the codeword of an ECC is usually represented in a systematic form consisting of the raw data and the parity information generated by encoding,the proposed architecture parallelizes the comparison of the data and that of the parity information. To further reduce the latency and complexity, in addition, a new butterfly-formed weight accumulator(BWA) is proposed for the efficient computation of the Hamming distance. Grounded on the BWA, the proposed architecture examines whether the incoming data matches the stored data if a certain number of erroneous bits are corrected. For a (40, 33) code, the proposed architecture reduces the latency and the hardware complexity by ∼32% and 9%, respectively, compared with the most recent implementation. Data comparison is widely used in computing systems to perform many operations such as the tag matching in a cache memory and the virtual-to-physical address translation in a translation look aside buffer (TLB). Because of such prevalence, it is important to implement the comparison circuit with low hardware complexity. Besides, the data comparison usually resides in the critical path of the components that are devised to increase the system performance

Keyword: *Data Comparison, Error-Correcting Codes (ECCs), Hamming Distance, Systematic Codes, Tag Matching*

1. Introduction

Data comparison circuit is a logic that has many applications in a computing system. For example, to check whether a piece of information is in a cache, the address of the information in the memory is compared to all cache tags in the same set that might contain that address. Error correction codes (ECC) are the one, most commonly used to protect standard memories and circuits, while more sophisticated codes are used in critical applications such as space. ECC are widely used to enhance the reliability and data integrity of memory structures in modern microprocessors. For example, caches on modern microprocessors are protected by ECC. If a memory structure is protected with ECC, a piece of data is encoded first and the entire codeword including the ECC check bits are written into the memory array. When the input data is loaded into the system, it has to be encoded and compared with the data stored in the memory and corrected if errors are detected to obtain the original data. Data comparison circuit is usually in the critical path of a pipeline stage because the result of the comparison determines the flow of the succeeding operations. When the memory array is protected by ECC, it exacerbates the criticality because of the added latency due to ECC logic. In the cache tag matching example, the cache tag directory must be accessed first. After the tag information is retrieved, it must go through ECC decoding and correction before the comparison operation can be performed. At the mean time, the corresponding data array is waiting for the comparison result to decide which way in the set to load the data from. The most recent solution for the matching problem is the direct compare method, which encodes the incoming data and then compares it with the retrieved data that has been encoded as well. Therefore, the method eliminates the complex decoding from the critical path. In performing the comparison, the method does not examine whether the retrieved data is exactly the same as the incoming data. Instead, it checks if the retrieved data resides in the error correctable range of the codeword corresponding to the incoming data. As the checking necessitates an additional circuit to compute the Hamming distance, i.e., the number of different bits between the two code words, the saturate adder (SA) was presented as a basic building block for calculating the Hamming distance. However, it does not consider an important fact that a practical ECC codeword is usually represented in a systematic form in which the data and parity bits are completely separated from each other. In addition, SA contributes to the increase of the entire circuit complexity as it always forces its output not to be greater than the number of detectable errors by more. In brief, we renovate the SA-based direct compare architecture to reduce the latency and hardware complexity by resolving the drawbacks. More specifically, we consider the characteristics of systematic codes in designing the proposed architecture and propose a low-complexity processing element that computes the Hamming distance faster. Therefore, the latency and the hardware complexity are decreased considerably compared with the SA based architecture. In the decode-and-compare architecture, the nbit retrieved codeword should first be decoded to extract the original k-bit tag. The extracted k-bit tag is then compared with the k-bit tag field of an incoming address to determine whether the tags are matched or not. As the retrieved codeword should go through the decoder before being compared with the incoming tag conventional decode-andcompare architecture and encode and compare architecture. To resolve the drawbacks of the decode-and-compare architecture, therefore, the decoding of a retrieved codeword is replaced with the encoding of an incoming tag in the encode-and-compare architecture. A k-bit incoming tag is first encoded to the corresponding n-bit codeword X and compared with an n-bit retrieved codeword Y. The comparison is to examine how many bits the two code words differ, not to check if the two code words are exactly equal to each other. For this, we compute the Hamming distance d between the two code words and classify the cases according to the range of d. In the SA-based architecture, the comparison of two code words is invoked after the incoming tag is encoded. Therefore, the critical path consists of a series of the encoding and the n-bit comparison. However, it did not consider the fact that, in practice, the ECC codeword is of a systematic form in which the data and parity parts are completely separated. As the data part of a systematic codeword is exactly the same as the incoming tag field, it is immediately available for comparison while the parity part becomes available only after the encoding is completed.

2. Proposed Methodology

The proposed design parallelizes the comparison of the data and which of the parity information. Error correction code permits data that's being read or transmitted to be checked for errors and, once necessary, corrected on the fly. It differs from parity-checking in those errors don't seem to be solely detected however additionally corrected. We will scale back the realm of the proposed System and additionally we will additionally increase the correcting code word length. We will perform the error correction for numerous code word lengths. During this project we tend to do constant operation for (16,8) and (40,33) codes additionally. A translation look a side buffer (TLB) could be a cache that memory management hardware uses to boost virtual address translation speed. The bulk of desktop, laptop, and server processors includes one or additional TLBs within the memory management hardware, and it's nearly continuously gift in any hardware that utilizes paged or divided virtual storage. The TLB is usually enforced as content-addressable memory (CAM). The CAM search key's the virtual address and therefore the search result's a physical address. If the requested address is gift within the TLB, the CAM search yields a match quickly and therefore the retrieved physical address will be wont to access memory. This is often known as a TLB hit. If the requested address isn't within the TLB, it's a miss, and therefore the translation income by trying up the page table in a very method known as a page walk. The page walk needs plenty of your time in comparison to the processor speed, because it involves reading the contents of multiple memory locations and mistreatment them to reason the physical address. Once the physical address is decided by the page walk, the virtual address to physical address mapping is entered into the TLB. Here's however it works for information storage:

- Once a unit of information is keep in RAM or peripheral storage, a code that describes the bit sequence within the word is calculated and stored in conjunction with the unit of information. For every 64-bit word, an additional seven bits square measure required to store this code.
- Once the unit of information is requested for reading, a code for the stored and about-to-be-read word is once more calculated mistreatment the initial algorithmic program. The new generated code is compared with the code generated once the word was stored.
- If the codes match, the information is freed from errors and is distributed.

• If the codes do not match, the missing or incorrect bits square measure determined through the code comparison and therefore the bit or bits square measure equipped or corrected.
• No try is created to correct the information that's still in storage. Eventually, it'll be overlaid by new information and, presumptuous the errors were transient; the inaccurate bits can "go away."
• Any error that recurs at constant place in storage once the system has been turned off and on once more indicate a permanent computer error and a message is distributed to a log or to a supervisor indicating the placement with the repeated errors. At the 64-bit word level, parity-checking and error correction code need constant range of additional bits. In general, error correction code will increase the reliability of any computing or telecommunications system (or a part of a system) while not adding abundant value. Reed-Solomon codes square measure normally implemented; they are ready to notice and restore "erased" bits additionally as incorrect bits.

2.1 Hamming Distance

In information theory, the performing distance between 2 strings of equal length is that the variety of positions at that the corresponding symbols area unit totally different. In our own way, it measures the minimum variety of substitutions needed to vary one string into the other, or the minimum variety of errors that might have transformed one string into the other. For binary strings a and b the Hamming distance is adequate the number of one’s in an exceedingly a XOR b. The mathematical space of length-n binary strings, with the Hamming distance, is thought because the Hamming cube; it's equivalent as a mathematical space to the set of distances between vertices in an exceedingly hypercube graph. One also can read a binary string of length n as a vector in Rn by treating every symbol within the string as a real coordinate; with this embedding, the strings are formed the vertices of associate n-dimensional hypercube, and also the Hamming distance of the strings is admire the Manhattan distance between the vertices. The Hamming Distance may be a number used to denote the distinguish between 2 binary strings. It’s a little portion of a broader set of formulas utilized in info analysis. Specifically, Hamming's formulas enable computers to detect and correcting error on their own. The Hamming Code earned Richard Hamming the Eduard Rheim Award of feat in Technology in 1996, 2 years before his death. Hamming's additions to info technology are utilized in such innovations as modems and compact discs.

Step 1: Ensure the 2 strings area unit of equal length. The hamming distance will solely be calculated between 2 strings of equal length. String 1: "1001 0010 1101" String 2: "1010 0010 0010".

Step 2: Compare the primary 2 bits in every string. If they're identical, record a "0" for that bit. If they're totally different, record a "1" for that bit. During this case, the first bit of both strings is "1," thus record a "0" for the primary bit.

Step 3: Compare every bit in succession and record either "1" or "0" as acceptable. String 1: "1001 0010 1101" String 2: "1010 0010 0010" Record: "0011 0000 1111".

Step 4: Add all ones and zeros within the record along to get the Hamming distance. Hamming distance = 0+0+1+1+0+ 0+0+0+1+1+1+1 = 6.

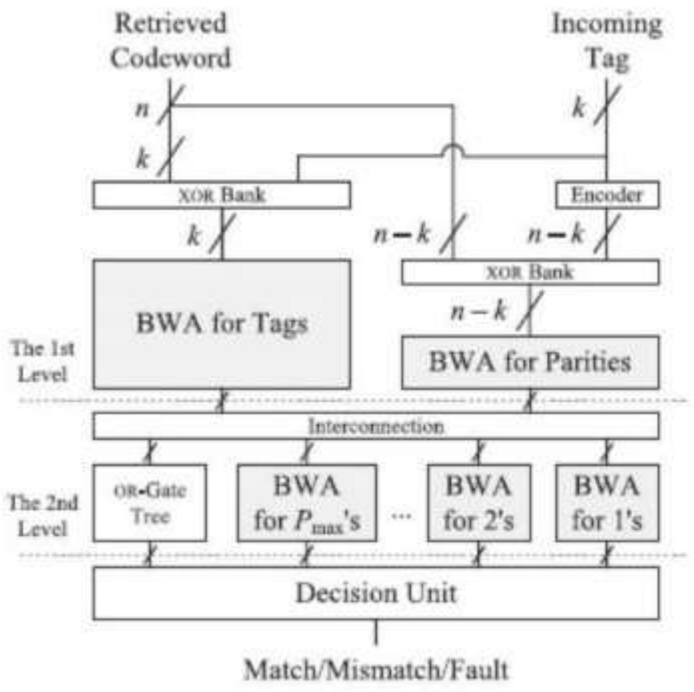
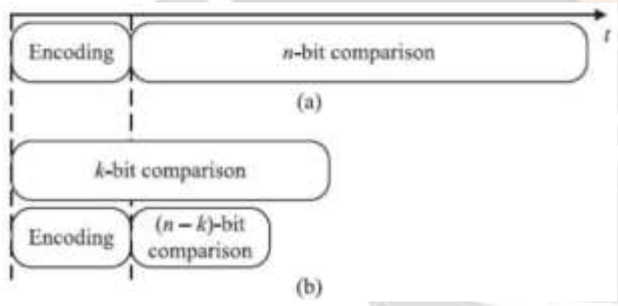


Fig -1 Proposed architecture optimized for systematic code words.

It contains multiple butterfly-formed weight accumulators (BWAs) proposed to improve the latency and complexity of the Hamming distance computation. The basic function of the BWA is to count the number of 1's among its input bits. This section presents a new architecture that can reduce the latency and complexity of the data comparison by using the characteristics of systematic codes as shown in Fig.1. In addition, a new processing element is presented to reduce the latency and complexity further as shown in Fig.2.



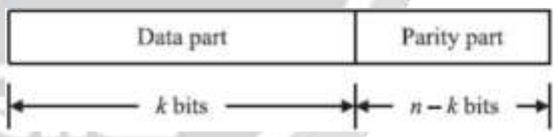


Fig -2 Timing diagram of the tag match in proposed architecture.

Fig.3. Systematic representation of an ECC codeword.

In practice, the ECC codeword is of a systematic form in which the data and parity parts are completely separated as shown in the above Fig.3.

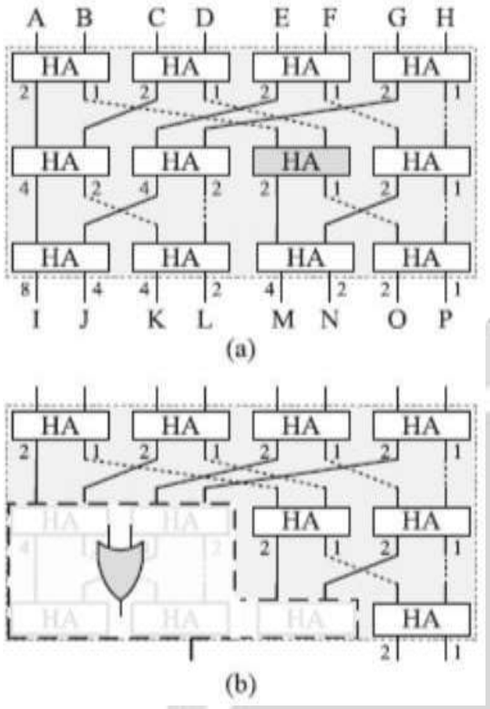


Fig.4. Proposed BWA.

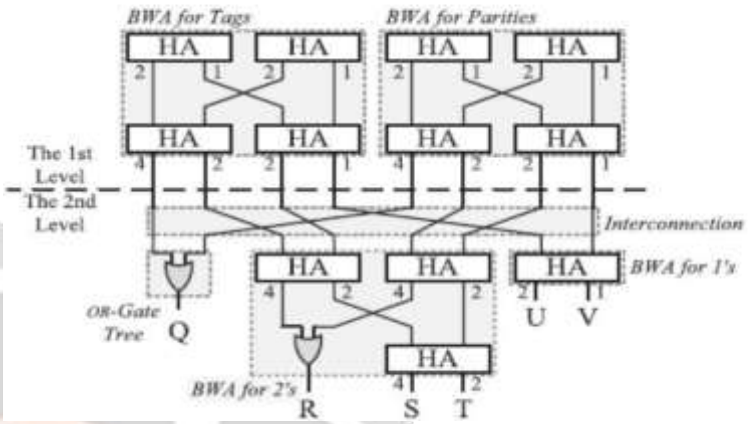
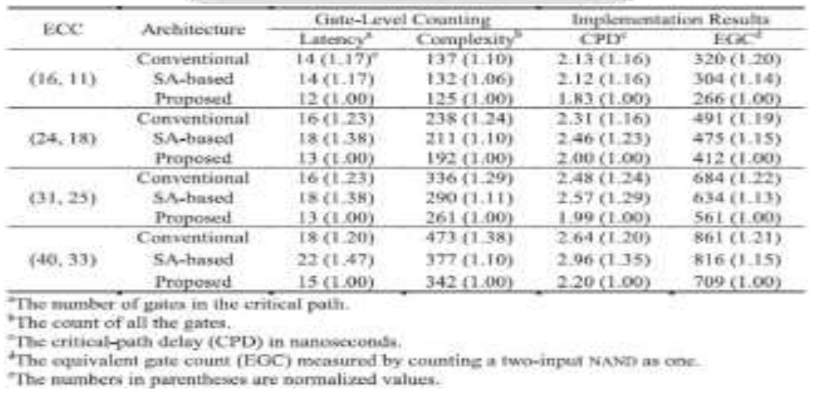


Fig.5. First and second level circuits for a (8,4) code.

3. Evaluation

For a set of four codes including the (31, 25) code quoted from, Table I shows the latencies and hardware complexities resulting from three architectures: 1) the conventional decode-and-compare; 2) the SA-based direct compare; and 3) the proposed ones. We measured the metrics at the gate-level first and then implemented the circuits in a 0.13-μm CMOS technology to provide more realistic results by deliberating some practical factors, e.g., gate sizing and wiring delays. In, the latency is measured from the time when the incoming address is completely encoded as shown in Fig.4. As the critical path starts from the arrival of the incoming address to a cache memory, the encoding delay must be, however, included in the latency computation the latency values in Table I are all measured in this way. Besides, critical-path delays in Table I are obtained by performing post layout simulations and equivalent gate counts are measured by counting a two-input NAND as one.

TABLE I: Comparison for Latency and Hardware Complexity



ECC	Architecture	Gate-Level Counting		Implementation Results	
		Latency[a]	Complexity[b]	CPD[c]	EGC[d]
(16, 11)	Conventional	14 (1.17)[e]	137 (1.10)	2.13 (1.16)	320 (1.20)
	SA-based	14 (1.17)	132 (1.06)	2.12 (1.16)	304 (1.14)
	Proposed	12 (1.00)	125 (1.00)	1.83 (1.00)	266 (1.00)
(24, 18)	Conventional	16 (1.23)	238 (1.24)	2.31 (1.16)	491 (1.19)
	SA-based	18 (1.38)	211 (1.10)	2.46 (1.23)	475 (1.15)
	Proposed	13 (1.00)	192 (1.00)	2.00 (1.00)	412 (1.00)
(31, 25)	Conventional	16 (1.23)	336 (1.29)	2.48 (1.24)	684 (1.22)
	SA-based	18 (1.38)	290 (1.11)	2.57 (1.29)	634 (1.13)
	Proposed	13 (1.00)	261 (1.00)	1.99 (1.00)	561 (1.00)
(40, 33)	Conventional	18 (1.20)	473 (1.38)	2.64 (1.20)	861 (1.21)
	SA-based	22 (1.47)	377 (1.10)	2.96 (1.35)	816 (1.15)
	Proposed	15 (1.00)	342 (1.00)	2.20 (1.00)	709 (1.00)

[a]The number of gates in the critical path.
[b]The count of all the gates.
[c]The critical-path delay (CPD) in nanoseconds.
[d]The equivalent gate count (EGC) measured by counting a two-input NAND as one.
[e]The numbers in parentheses are normalized values.

As shown in Table I, the proposed architecture is effective in reducing the latency as well as the hardware complexity even with considering the practical factors. Note that the effectiveness of the proposed architecture over the SA-based one in shortening the latency gets larger as the size of a codeword increases as shown in Fig.5. The reason is as follows. The latencies of the SA-based architecture and the proposed one are dominated by SAs and HAs, respectively. As the bit-width doubles, at least one more stage of SAs or HAs needs to be added. Since the critical path of a HA consists of only one gate while that of a SA has several gates, the proposed architecture achieves lower latency than its SA-based counterpart, especially for long code words.

4.Simulation Result

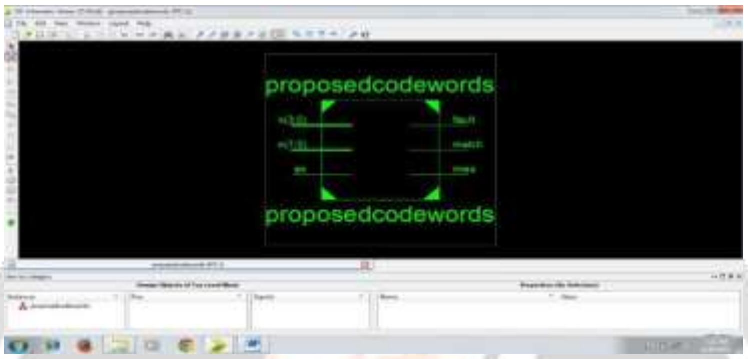


Fig.6. RTL Schematic

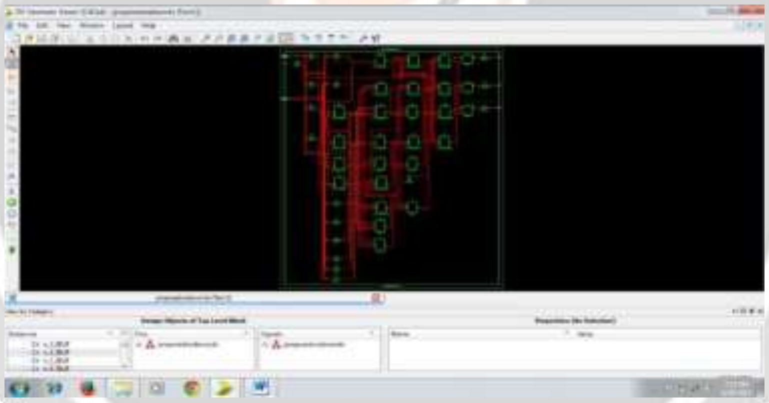


Fig.7. Internal Architecture of schematic

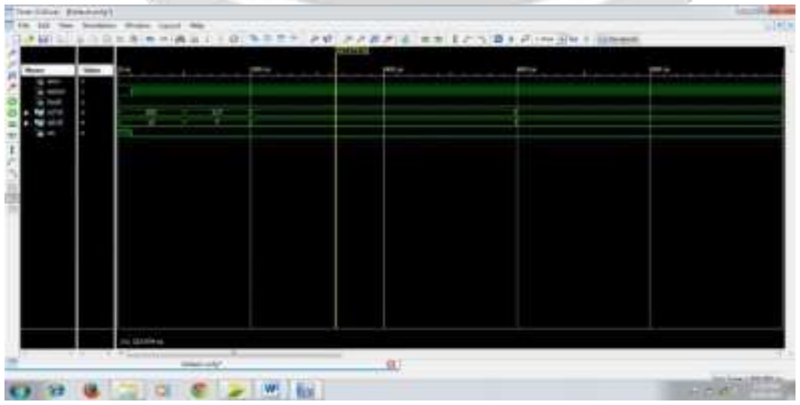


Fig.8. Simulation Result

5. CONCLUSIONS

In this process, we formulate the DMC technique to assure the consistency in memory. The proposed protection code utilizes decimal algorithm to detect errors, so that more errors were detected and corrected. To reduce the hardware complexity and latency, a new architecture has been presented for matching the data protected with an ECC. To reduces the latency; the comparison of the data is parallelized with the encoding process that generates the parity information. The parallel operations are enabled based on the fact that the systematic codeword has separate fields for the data and parity. In addition, an efficient processing architecture has been presented to further minimize the latency and complexity. Consequently a sensible reduction in power is accomplished with the proposed devise.

6. REFERENCES

[1] Byeong Yong Kong, Jihyuck Jo, Hyewon Jeong, Mina Hwang, Soyoung Cha, Bongjin Kim, and In-Cheol Park, "Low-Complexity Low-Latency Architecture for Matching of Data Encoded with Hard Systematic Error-Correcting Codes", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 22, No. 7, July 2014.
[2] 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.
[3] 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 the microprocessor chip for the z-Enterprise system," IEEE J. Solid-State Circuits, vol. 47, no. 1, pp. 151–163, Jan. 2012.
[4] H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, and T. Motokurumada, S. Okada, H. Yamashita, and Y. Satsukawa, "A 1.3 GHz fifth generation SPARC64 microprocessor," in IEEE ISSCC. Dig. Tech. Papers, Feb. 2003, pp. 246–247.
[5] 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.