Low-Complexity Low-Latency Architecture for Matching Of Data Encoded With Hard Systematic Error-Correcting Codes

***Shashi vardhan & **B Vasunaik**

*M-Tech Student Ganapathy Engineering College, Rangasaipet, Hunter Road, Warangal

**Associate Professor ECE Dept Ganapathy Engineering College, Rangasaipet, Hunter Road, Warangal

WWW.VASU523@GMAIL.COM

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 code word 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.*

I. INTRODUCTION

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, e.g., caches and TLBs, whose outputs determine the flow of the succeeding operations in a pipeline. The circuit, therefore, must be designed to have as low latency as possible, or the components will be disqualified from serving as accelerators and the overall performance of the whole system would be severely deteriorated. As recent computers employ error-correcting codes (ECCs) to protect data and improve reliability[1]–[5], complicated decoding procedure, which must precede the data comparison, elongates the critical path and exacerbates the complexity overhead. Thus, it becomes much harder to meet the above design constraints. Despite the need for sophisticated designs as described, the works that cope with the problem are not widely known in the literature since it has been usually treated within industries for their products. Recently, however, [6] triggered the attraction of more and more attentions from the academic field. The most recent solution for the matching problem is the direct compare method [6], 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

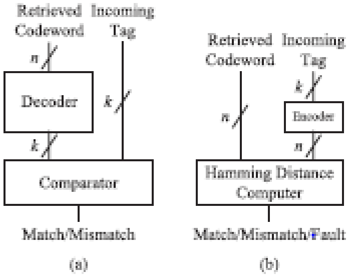

Fig. 1. (a) Decode-and-compare architecture and (b) encode-and-compare architecture.

incoming data. Instead, it checks if the retrieved data resides in the error correctable range of the code word 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 codewords, the saturate adder (SA) was presented in [6] as a basic building block for calculating the Hamming distance. However, [6] did not consider an important fact that may improve the effectiveness further, a practical ECC codeword is usually represented in a systematic form in which the data and parity parts are completely separated from each other [7]. In addition, as the SA always forces its output not to be greater than the number of detectable errors by more than one, it contributes to the increase of the entire circuit complexity. In this brief, we renovate the SA-based direct compare architecture to reduce the latency and hardware complexity by resolving the aforementioned 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 even compared with the SA based architecture. The rest of this brief is organized as follows. Section II reviews previous works. The proposed architecture is explained in Section III, and evaluated in Section IV. Finally, concluding remarks are made in Section V.

II. PREVIOUS WORKS

This section describes the conventional decode-and-compare architecture and the encode-and-compare architecture based on the direct compare method. For the sake of concreteness, only the tag matching performed in a cache memory is discussed in this brief, but the proposed architecture can be applied to similar applications without loss of generality.

A. *Decode-and-Compare Architecture*

Let us consider a cache memory where a *k*-bit tag is stored in the form of an *n*-bit code word after being encoded by a (*n*, *k)* code. In the decode-and-compare architecture depicted in Fig. 1(a), the *n*-bit retrieved code word 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 code word should go through the decoder before being compared with the incoming tag, the critical path is too long to be employed in a practical cache system designed for high-speed access. Since the decoder is one of the most complicated processing elements, in addition, the complexity overhead is not negligible.

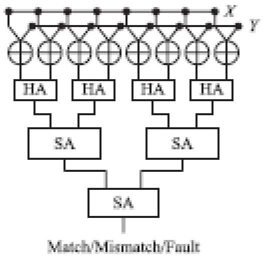

Fig. 2. SA-based architecture supporting the direct compare method [6].

B. Encode-and-Compare Architecture

Note that decoding is usually more complex and takes more time than encoding as it encompasses a series of error detection or syndrome calculation, and error correction [7]. The implementation results in [8] support the claim. To resolve the drawbacks of the decode-and-compare architecture, therefore, the decoding of a retrieved code word is replaced with the encoding of an incoming tag in the encode-and-compare architecture More precisely, a *k*-bit incoming tag is first encoded to the corresponding *n*-bit codeword *X* and compared with an *n*-bit retrieved codeword *Y* as shown in Fig. 1(b). The comparison is to examine how many bits the two codewords differ, not to check if the two codewords are exactly equalto each other. For this, we compute the Hamming distance *d* between the two codewords and classify the cases according to the range of *d*.Let *t*max and *r*max denote the numbers of maximally correctable and detectable errors, respectively. The cases are summarized as follows.

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

2) If $0 < d \leq t\text{max}$, X will match Y provided at most *t*max errors in Y are corrected.

3) If $t\text{max} < d \leq r\text{max}$, Y has detectable but uncorrectable errors.

In this case, the cache may issue a system fault so as to make the central processing unit take a proper action.

4) If $r\text{max} < d$, X does not match Y .

Assuming that the incoming address has no errors, we can regard the two tags as matched if *d* is in either the first or the second ranges. In this way, while maintaining the error-correcting capability, the architecture can remove the decoder from its critical path at the cost of an encoder being newly introduced. Note that the encoder is, ingeneral, much simpler than the decoder, and thus the encoding cost is significantly less than the decoding cost. Since the above method needs to compute the Hamming distance, [6] presented a circuit dedicated for the computation. The circuit shown in Fig. 2 first performs XOR operations for every pair of bits in *X* and *Y* so as to generate a vector representing the bitwise difference of the two code words. The following half adders (HAs) are used to count the number of 1's in two adjacent bits in the vector. The numbers of 1's are accumulated by passing through the following SA tree. In the SA tree, the accumulated value *z* is saturated to *r*max + 1if it exceeds *r*max. More precisely, given inputs *x* and *y*, *z* can be expressed as follows:

$$z = \begin{cases} x + y, & \text{if } x + y \leq r_{\max} \\ r_{\max} + 1, & \text{otherwise.} \end{cases} \qquad (1)$$

The final accumulated value indicates the range of *d*. As the compulsory saturation necessitates additional logic circuitry, the complexity of a SA is higher than the conventional adder.

III. PROPOSED ARCHITECTURE

This section presents a new architecture that can reduce the latency and complexity of the data comparison by using the characteristics of

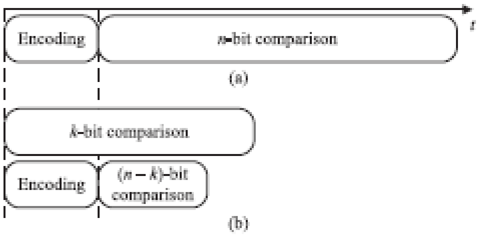

Fig. 3. Timing diagram of the tag match in (a) direct compare method [6] and (b) proposed architecture.

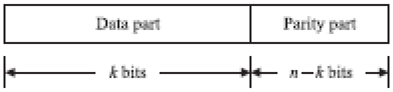

Fig. 4. Systematic representation of an ECC codeword.

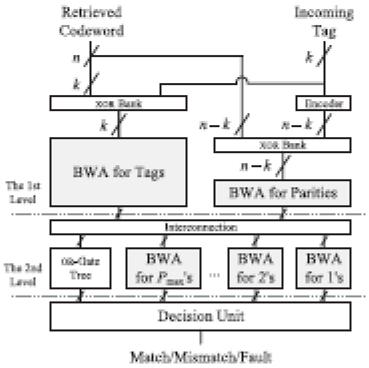

Fig. 5. Proposed architecture optimized for systematic codewords.

systematic codes. In addition, a new processing element is presented to reduce the latency and complexity further.

A. Datapath Design for Systematic Codes

In the SA-based architecture [6], 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 as shown in Fig. 3(a). However, [6] did not consider the fact that, in practice, the ECC code word is of a systematic form in which the data and parity parts are completely separated as shown in Fig. 4. As the data part of a systematic code word 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.Grounded on this fact, the comparison of the *k*-bit tags can be started before the remaining (*n–k)*-bit comparison of the parity bits. In the proposed architecture, therefore, the encoding process to generate the parity bits from the incoming tag is performed in parallel with the tag comparison, reducing the overall latency as shown in Fig. 3(b).

B. Architecture for Computing the Hamming Distance

The proposed architecture grounded on the datapath design is shown in Fig. 5. 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. It consists of multiplestages of HAs as shown in Fig. 6(a), where each output bit of a HA

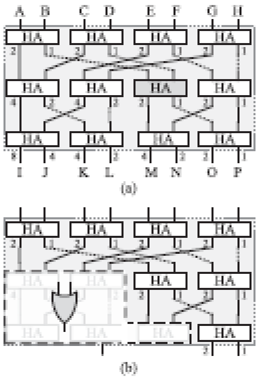

Fig. 6. Proposed BWA. (a) General structure and (b) new structure revised for the matching of ECC-protected data. Note that sum-bit lines are dotted for visibility.

is associated with a weight. The HAs in a stage are connected in a butterfly form so as to accumulate the carry bits and the sum bits of the upper stage separately. In other words, both inputs of a HA in a stage, except the first stage, are either carry bits or sum bits computed in the upper stage. This connection method leads to a property that if an output bit of a HA is set, the number of 1's among the bits in the paths reaching the HA is equal to the weight of the output bit.In Fig. 6(a), for example, if the carry bit of the gray-colored HA is set, the number of 1's among the associated input bits, i.e., A, B, C, and D, is 2. At the last stage of Fig. 6(a), the number of 1's among the input bits, *d*, can be calculated as

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

Since what we need is not the precise Hamming distance but the range it belongs to, it is possible to simplify the circuit. When *r*max = 1, for example, two or more than two 1's among the input bits can be regarded as the same case that falls in the fourth range. In that case, we can replace several HAs with a simple OR-gate tree as shown in Fig. 6(b). This is an advantage over the SA that resorts to the compulsory saturation expressed in (1). Note that in Fig. 6, there is no overlap between any pair of two carry-bit lines or any pair of two sum-bit lines. As the overlaps exist only between carry-bit lines and sum-bit lines, it is not hard to resolve overlaps in the contemporary technology that provides multiple routing layers no matter how many bits a BWA takes.We now explain the overall architecture in more detail. Each XOR stage in Fig. 5 generates the bitwise difference vector for either data bits or parity bits, and the following processing elements count the number of 1's in the vector, i.e., the Hamming distance. Each BWA at the first level is in the revised form shown in Fig. 6(b), and generatesan output from the OR-gate tree and several weight bits from the HA trees. In the interconnection, such outputs are fed into their associated processing elements at the second level. The output of the OR-gate tree is connected to the subsequent OR-gate tree at the second level, and the remaining weight bits are connected to the second level BWAs according to their weights. More precisely, the bits of weight *w* are connected to the BWA responsible for *w*-weight inputs. Each BWA at the second level is associated with a weight of a power of two that is less than or equal to *P*max, where *P*max is the largest power of two that is not greater than *r*max + 1. As the weight bits associated with the fourth range are all ORed in the revised BWAs, there is no need to deal with the powers of two that are larger than *P*max.For example, let us consider a simple (8, 4) single-error correction not drawn in Fig. 7 for the sake of simplicity. Since *r*max = 2, *P*max = 2 and there are only two BWAs dealing with weights 2 and 1 at the second level. As the bits of weight 4 fall in the fourth range, they are ORed. The remaining bits associated with weight 2 or 1 are connected to their corresponding BWAs. Note that the interconnection induces no hardware complexity, since it can be achieved by a bunch of hard wiring. Taking the outputs of the preceding circuits, the decision unit finally determines if the incoming tag matches the retrieved codeword by considering the four ranges of the Hamming distance. The decision unit is in fact a combinational logic of which functionality is specified by a truth table that takes the outputs of the preceding circuits as inputs. For the (8, 4) code that the corresponding first and secondlevel circuits are shown in Fig. 7, the truth table for the dec

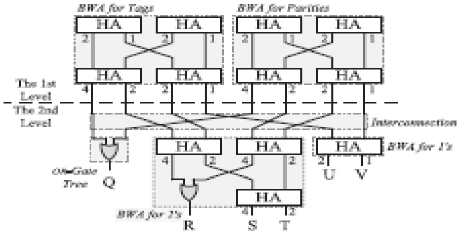

Fig. 7. First and second level circuits for a (8, 4) code.

TABLE I
TRUTH TABLE OF THE DECISION UNIT FOR A (8, 4) CODE

Q OR R OR S	T	U	V	Decision
0	0	0	x	Match
	0	1	x	Fault
	1	0	0	Fault
	1	0	1	Mismatch
	1	1	x	Mismatch
1	x	x	x	Mismatch

double-error detection code. The corresponding first and second level circuits are shown in Fig. 7. Note that the encoder and XOR banks are ision unit is described in Table I. Since *U* and *V* cannot be set simultaneously,such cases are implicitly included in do not care terms in Table I. *C. General Expressions for the Complexity and the Latency* The complexity as well as the latency of combinational circuits heavily depends on the algorithm employed. In addition, as the complexity and the latency are usually conflicting with each other, it is unfortunately hard to derive an analytical and fully deterministic equation that shows the relationship between the number of gates and the latency for the proposed architecture and also for the conventional SA-based architecture. To circumvent the difficulty in analytical derivation, we present instead an expression that can be used to estimate the complexity and the latency by employing some variables for the nondeterministic parts. The complexity of the proposed architecture, *C*, can be expressed as

C = *C*XOR+ *C*ENC+ *C*BWA*(k)* + *C*BWA*(n− k)*+ *C*2nd + *C*DU

≤ *n* + *C*ENC + 2*C*BWA*(n)* + *C*DU

where *C*XOR, *C*ENC, *C*2nd, *C*DU, and *C*BWA*(n)* are the complexities of XOR banks, an encoder, the second level circuits, the decision unit, and a BWA for *n* inputs, respectively. Using the recurrence relation,

TABLE II
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.

*C*BWA*(n)* can be calculated as

*C*BWA*(n)* = *C*BWA *(_n/*2*_)* + *C*BWA *(_n/*2*_)* + 2 *_n/*2*_* (4)

where the seed value, *C*BWA(1), is 0. Note that when *a* + *b* = *c*,

*C*BWA*(a)* + *C*BWA*(b)* ≤ *C*BWA*(c)* holds for all positive integers *a*, *b*, and *c*. Because of the inequality and the fact that an OR-gate tree for *n* inputs is always simpler than a BWA for *n* inputs, both *C*BWA*(k)* + *C*BWA*(n –k)* and *C*2nd are bounded by *C*BWA*(n)*.The latency of the proposed architecture, *L*, can be expressed as

$$\begin{aligned} L \leq\ & \max\left[L_{\mathrm{XOR}} + L_{\mathrm{BWA}}(k), L_{\mathrm{ENC}} + L_{\mathrm{XOR}} + L_{\mathrm{BWA}}(n-k)\right] \\ & + L_{\mathrm{2nd}} + L_{\mathrm{DU}} \\ \leq\ & \max\left(1 + \lceil \log_2 k \rceil, L_{\mathrm{ENC}} + 1 + \lceil \log_2 (n-k) \rceil\right) \\ & + \lceil \log_2 n \rceil + L_{\mathrm{DU}} \qquad (5) \end{aligned}$$

where *L*XOR, *L*ENC, *L*2nd, *L*DU, and *L*BWA*(n)* are the latencies of an XOR bank, an encoder, the second level circuits, the decision unit, and a BWA for *n* inputs, respectively. Note that the latencies of the OR-gate tree and BWAs for *x* ≤ *n* inputs at the second level are all bounded by _ log2 *n* _ . As one of BWAs at the first level finishes earlier than the other, some components at the second level may start earlier. Similarly, some BWAs or the OR-gate tree at the second level may provide their output earlier to the decision unit so that the unit can begin its operation without waiting for all of its inputs. In such cases, L2nd and LDU can be partially hidden by the critical path of the preceding circuits, and L becomes shorter than the given expression.

IV. EVALUATION

For a set of four codes including the (31, 25) code quoted from [6],Table II 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 [6], the latency is measured from the time when the incoming address is completely encoded. 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 latencycomputation. The latency values in Table II are all measured in this way. Besides, critical-path delays in Table II are obtained by performing postlayout simulations and equivalent gate counts are measured by counting a two-input NAND as one.As shown in Table II, 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. The reason is as follows. The latencies of the SA-based architecture and the proposedone 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 lowerlatency than its SA-based counterpart, especially for long codewords.

V. CONCLUSION

To reduce the latency and hardware complexity, a new architecture has been presented for matching the data protected with an ECC. The proposed architecture examines whether the incoming data matchesthe stored data if a certain number of erroneous bits are corrected. To reduce the latency, the comparison of the data is parallelized with theencoding process that generates the parity information. The paralleloperations 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 thelatency and complexity. As the proposed architecture is effective in reducing the latency as well as the complexity considerably, it can be regarded as a promising solution for the comparison of ECC-protected data. Though this brief focuses only on the tag match of a cachememory, the proposed method is applicable to diverse applications that need such comparison.

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, o. 4, pp. 846–852, Apr. 2007.

[2] J. D. Warnock, Y.-H. Chan, S. M. Carey, H. Wen, P. J. Meaney, G. Gerwig, . 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 ircuits*, vol. 47, no. 1, pp. 151–163, Jan. 2012.

[3] 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 *IEEEISSCC. Dig. Tech. Papers*, Feb. 2003, pp. 246–247.

[4] 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.

[5] AMD Inc. (2010). *Family 10h AMD Opteron Processor Product Data Sheet*, Sunnyvale, CA, USA [Online]. Available: http://support.amd.com/us/Processor_TechDocs/40036.pdf

Authors profile : - shashi vardhan completed his b tech and pursuing m tech in ganapathi engineering college

B.vasunaik completed his B TECH in E&I 1999 from kits Warangal and completed his M TECH in DSCE from vaagdevi college of engineering 2011, presently working as ASSOCIATE Professor in Ganapathi engineering college Warangal WWW.VASU523@GMAIL.COM