IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 10, Issue 4, Ver. II (Jul - Aug .2015), PP 28-34
www.iosrjournals.org

Synthesis of Data Encoded With Error Correcting Codes Using BWA Architecture

Steffi Philip Mulamoottil[1] Dr.E.Nagabhooshanam[2] Nimmagadda Ravali [3]
1.M.Tech VLSI System Design Student, Sridevi Womens Engineering College Hyderabad
2. HOD ECE Department ,Sridevi Womens Engineering College Hyderabad
3.Assistant Professor, Sridevi Womens Engineering College Hyderabad

I. Introduction

Data comparison is widely used in computing systems to perform many operations such as tag matching in cache memory, virtual to physical address translation in TLB. Because of such prevalence it is important to implement the comparison circuit with low hardware complexity. Data Fix has developed a series of algorithms and programs that utilize token-based matching in conjunction with various fuzzy-matching functions. For a given block of source data each token is compared to records from the target dataset. The data comparison usually resides in the critical path of the components that are devised to increase system performance. For example caches and TLBs whose outputs determine the flow of succeeding operations in a pipeline. Therefore the circuit must be designed to have as low latency as possible as latency describes the time interval between stimulation and response. Besides of not having low latency, the components will be disqualified from serving as accelerators and overall performance of the whole system would be severely deteriorated. Recently computers employ error correcting codes to protect data and improve reliability. Complicated decoding procedure precede the data comparison which must elongates the critical path and exacerbates the complexity overhead. Thus it seems to be harder to meet the above design constraints. Numerous algorithms have been developed to allow for so-called 'fuzzy' matching, including soundex, NYSIIS, Jaro-Winkler, and many different types of character transposition functions. While these approaches are useful for matching specific fields they are not capable of accurately comparing larger blocks of data.

The most recent solution for matching problem is the direct compare method. This method encodes the incoming data and then compared it with retrieved data that has been encoded as well. Therefore this method eliminates the complicated decoding procedure from critical path as it is more complex and consumes more time. While performing comparison method, it 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. While performing checking, an additional circuit for computing hamming distance is necessary. Hamming distance provides the number of different bits between two code words and for computing hamming distance saturate adder is considered as a building block. In addition SA forces its output not to be greater than the number of detectable errors by more than one. As ECC detects double and corrects single bit error and is represented in a systematic form in which data and parity parts are completely separated.

In this brief we renovate SA based direct compare architecture to reduce latency and hardware complexity by resolving the above mentioned drawbacks. More specifically we consider the error correcting codes in systematic form while designing the proposed architecture and provides a low complexity processing element that computes the hamming distance faster. Therefore latency and hardware complexity are decreased when compared with SA based architecture.

The rest of the paper is organised as follows. Section 2 describes the previous works. Section 3 describes the proposed architecture and section 4 describes the evaluation part. Finally conclusion is described in section 5.

II. Previous Works

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

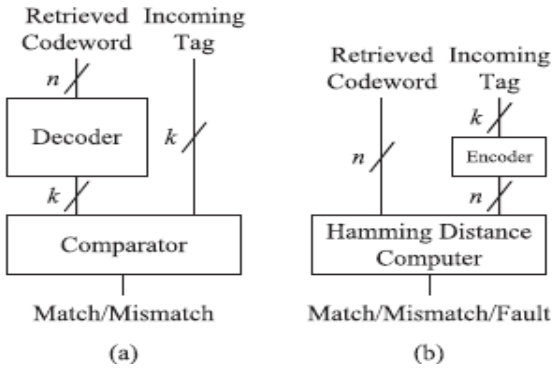

Fig 1. (a) Decode-and-compare architecture and (b) encode-and-compare architecture.

A. Decode-and-compare architecture

In decode and compare architecture the n-bit retrieved code word is first decoded to extract the original k-bit tag. The extracted k-bit tag is then compared with the k-bit tag field of the incoming address to determine whether the tags are matched or not as shown in Fig. 1(a). .As the retrieved codeword 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.

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. The implementation results support the claim. 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 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 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*.

Let t_{max} and r_{max} denote the numbers of maximally correctable and detectable errors, respectively. The cases are summarized as follows.

Condition	Result
1)d=0	X matches Y exactly
2)$0 \le d \le t_{max}$	X will match Y provided at the most t_{max} errors in Y are corrected
3)$t_{max} < d \le r_{max}$	Y has detectable but uncorrectable errors.
4)$r_{max} < d$	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, in general, 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, presented a circuit dedicated for the computation.

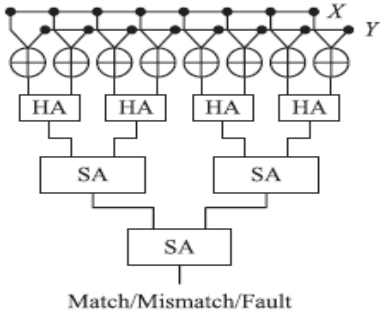

Fig 2. SA-based architecture supporting the direct compare method

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 + 1 if it exceeds *r*max. More precisely, given inputs *x* and *y*, *z* can be expressed as follows:

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

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. Butterfly Weight Accumulator (BWA) Architecture

This section describes a new architecture that can reduce the latency and complexity of the data comparison by using the characteristics of systematic codes. In addition, a new processing element is presented in this brief to reduce latency and complexity further

.

A. Datapath Design for Systematic Codes

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 as shown in Fig. 3(a).

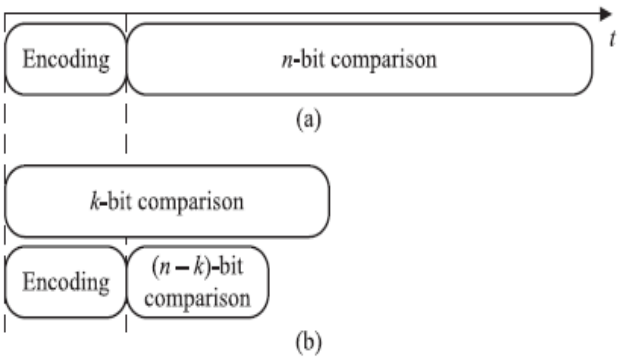

Fig. 3. Timing diagram of the tag match in (a) direct compare method and (b) proposed architecture.

However,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 shown in Fig. 4.

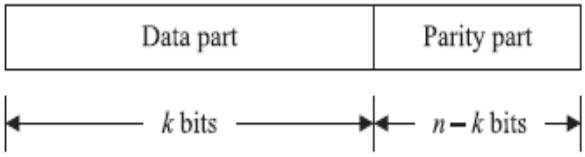

Fig. 4. Systematic representation of an ECC codeword.

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. 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 data path design is shown in Fig. 5. The proposed architecture contains multiple butterfly-formed weight accumulators (BWAs) proposed to improve the latency and complexity of the Hamming distance computation.

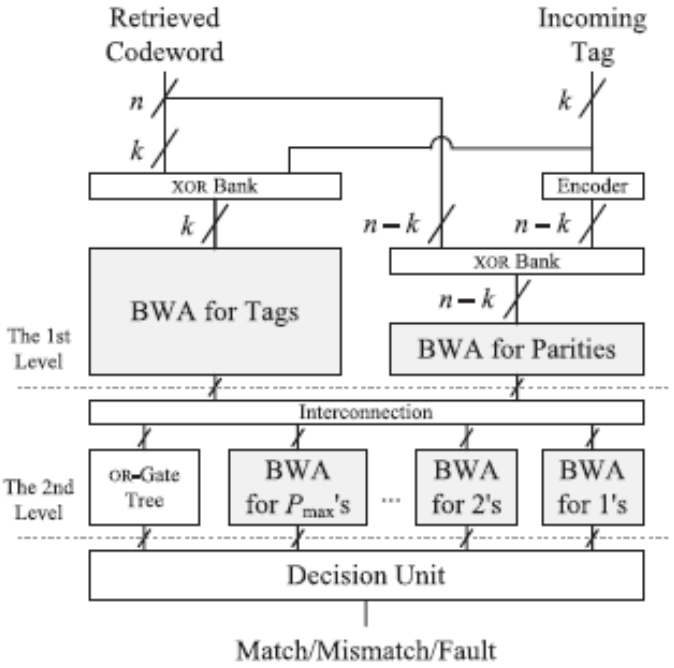

Fig.5 : Proposed architecture optimized for systematic code words

The basic function of the BWA is to count the number of 1's among its input bits. It consists of multiple stages of Half Adders as shown in Fig. 6(a), where each output bit of a HA 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-coloured 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

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

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).

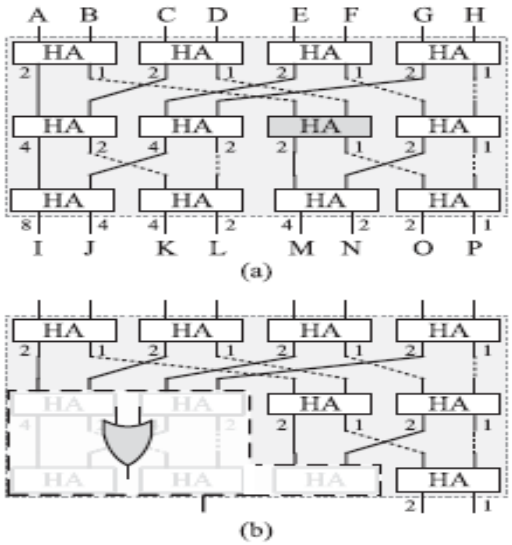

Fig. 6. Proposed BWA. (a) General structure and (b) new structure revised for the matching of ECC-protected data. Note that sum-bit lines are dotted for visibility

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 generates an 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 double-error detection code. The corresponding first and second level circuits are shown in Fig. 7. Note that the encoder and XOR banks are not drawn in Fig. 7 for the sake of simplicity.

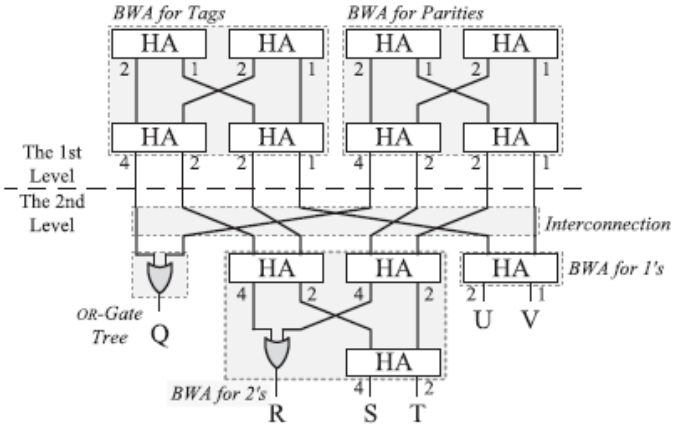

Fig. 7. First and second level circuits for a (8, 4) code.

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 second level circuits are shown in Fig. 7, the truth table for the decision 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.

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

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

$$\begin{aligned} C &= C_{\mathrm{XOR}} + C_{\mathrm{ENC}} + C_{\mathrm{BWA}}(k) + C_{\mathrm{BWA}}(n-k) + C_{\mathrm{2nd}} + C_{\mathrm{DU}} \\ &\le n + C_{\mathrm{ENC}} + 2C_{\mathrm{BWA}}(n) + C_{\mathrm{DU}} \end{aligned} \quad (3)$$

where C_{XOR}, C_{ENC}, $C_{2\mathrm{nd}}$, C_{DU}, and $C_{\mathrm{BWA}}(n)$ 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, $C_{\mathrm{BWA}}(n)$ can be calculated as

$$C_{\mathrm{BWA}}(n) = C_{\mathrm{BWA}}(\lfloor n/2 \rfloor) + C_{\mathrm{BWA}}(\lceil n/2 \rceil) + 2\lfloor n/2 \rfloor \quad (4)$$

where the seed value, $C_{\mathrm{BWA}}(1)$, is 0. Note that when $a + b = c$, $C_{\mathrm{BWA}}(a) + C_{\mathrm{BWA}}(b) \le C_{\mathrm{BWA}}(c)$ 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 $C_{\mathrm{BWA}}(k) + C_{\mathrm{BWA}}(n-k)$ and $C_{2\mathrm{nd}}$ are bounded by $C_{\mathrm{BWA}}(n)$. The latency of the proposed architecture, L, can be expressed as

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

where L_{XOR}, L_{ENC}, $L_{2\mathrm{nd}}$, L_{DU}, and $L_{\mathrm{BWA}}(n)$ 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 $x \le n$ inputs at the second level are all bounded by $[\log_2 n]$. 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. Proposed BWA Architecture

The above BWA architecture describes the operation of (8,4)code. The same architecture with some extensions is used to built the proposed architecture. Proposed architecture describes the analysis of (16,11),(24,18)and(40,33)code. The above mentioned (8,4) code takes only 8 bits of code words. Besides, the proposed architecture increases the bit length of the code words so that we can process more information.

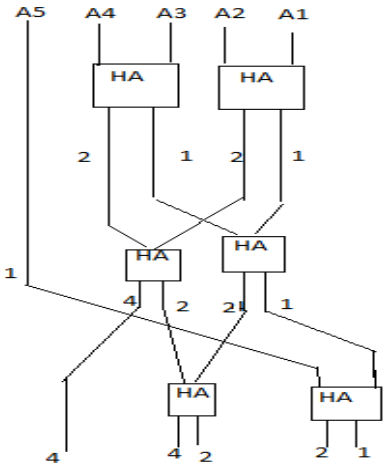

Fig 8(a): BWA For Parity of (16,11)code

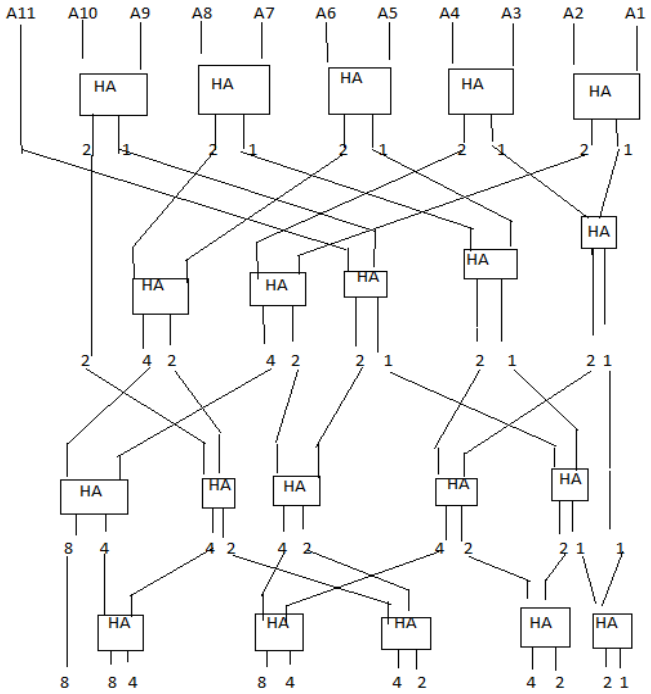

Fig 8(b) : BWA For Incoming tag of (16,11)code

The above figure shows the BWA architecture of (16,11) codeword with 11 bits of incoming tag information and 5 bits of parity information. Likewise we can draw architecture for (24,18) and (40,33).

V. Evaluation

For a set of four codes including the (31,25) code,Table 2 shows the latencies and hardware complexities resulting from three architectures:: 1)the decode-compare 2)the SA based direct compare and the 3) proposed ones. When measured the metrics at the gate level first and then implemented the circuits in CMOS technology to provide more realistic results by deliberating some practical factors e.g. gate sizing and wiring delays. The latency is measured from th 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 latency computation . The latency values in Table 2 are all measured in this way .Besides critical path delays in table 2 are obtained by performing post layout simulations and equivalent gate counts are measured by counting a two –input NAND gate as one.

As shown in table 2 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 the code word increases .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 atleast one more stages of the SAs or HAs needs to be added.Sin ce the critical path of the HA consists of only one gate while that of SA has several gates,the proposed architecture achieves lower latency than its SA based counterpart,especially for long codewords.

VI. Conclusion

To reduce latency and hardware complexity a new architecture has been presented in this brief for matching the data protected with an ECC.The proposed architecture examines whether the incoming data matches the stored data if a certain number of erroneous bits are corrected. To reduce latency the comparison of the data is parallelised with the encoding process that generates parity information.In addition an efficient processing architecture has been presented to further minimize the latency and complexity .The proposed architecture is effective in reducing the latency as well as the complexity . Though this brief focuses oly on tag matching in cache memory , the proposed method is applicable to diverse applications that need such comparison.