Power Efficient Error Detection Code Architecture for Encoded Data Matching

1 . Kathroju Sindhuja,M.Tech (VLSI&ES),2 .M.S Shyam, M.Tech (VLSI&ES), Assistant Professor ,
1,2. ECE Department,ST.MARY'S College of Engineering & Technology(Formely ASEC)

Abstract : AS CMOS technology scales down to nano scale and memories are combined with an increasing number of electronic systems, the soft error rate in memory cells is rapidly increasing. Although single bit upset is a major concern about memory reliability, multiple cell upsets (MCUs) have become a serious reliability concern in some memory applications which requires error correction. In order to make memory cells as fault-tolerant as possible, some error correction codes (ECCs) have been widely used to protect memories. For example, the Bose Chaudhuri Hocquenghem codes, Reed–Solomon codes, and punctured difference set (PDS) codes have been used to deal with MCUs in memories. But these codes require more area, power overheads since the encoding and decoding circuits are more complex in these complicated codes. In this project due to introduction of butterfly-formed weight accumulator (BWA) block to enhance the performance with less power consumption is proposed for efficient error correction.
KEYWORDS: Error correction codes, BWA, XILINX ISE, VERILOG

I.INTRODUCTION

Error detection and correction (EDAC) techniques are used to ensure that data is correct and has not been corrupted, either by hardware failures or by noise occurring during transmission or a data read operation from Memory. There are many different error correction codes in existence. The reason for the different codes being used in different applications has to do with the historical development of the data storage, the types of data errors occurring, and the overhead associated with each of the error detection techniques. The basic concept of error detection and correction method is as follow 1.Networks must be able to transfer data from one system to another without data can be corrupted during transmission.

For reliable communication, errors must be detected and corrected. Any error-correcting code can be used for error detection and correction. Error-correcting code (ECC) or forward error correction (FEC) code is a system of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors were introduced, either during the process of transmission, or on storage. Since the receiver does not have to ask the sender for retransmission of the data, a back-channel is not required in forward error correction, and it is therefore suitable for simplex communication such as broadcasting. Error-correcting codes are frequently used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks, and RAM. The general idea for achieving error detection and correction is to add some redundancy to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted. . Error-detection and correction schemes can be either systematic or non-systematic: In a systematic scheme, the transmitter sends the original data, and attaches a fixed number of check bits. which are derived from the data bits by some deterministic algorithm. If only error detection is required,

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 lookaside 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 reliabil-ity [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.

II EXISTING ARCHITECTURE

This section describes the conventional decode-and-compare archi-tecture 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 codeword after being encoded by a (n, k) code. In the decode-and-compare architecture depicted in Fig. 1(a), the n-bit 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, 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.

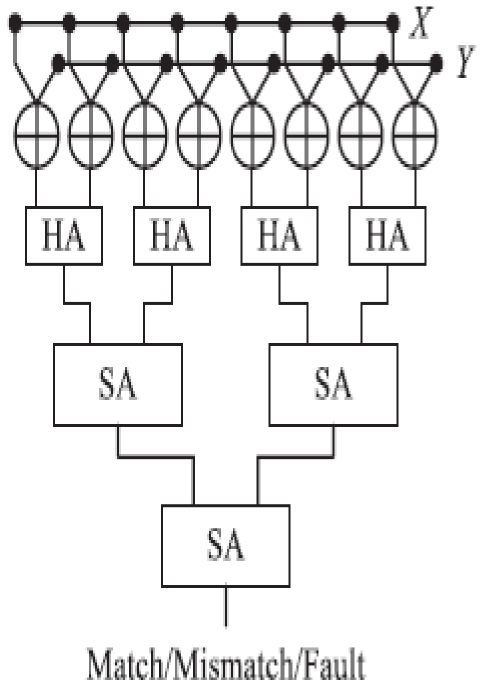


Fig. 1. SA-based architecture supporting the direct compare method

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 implemen-tation results in [8] 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).

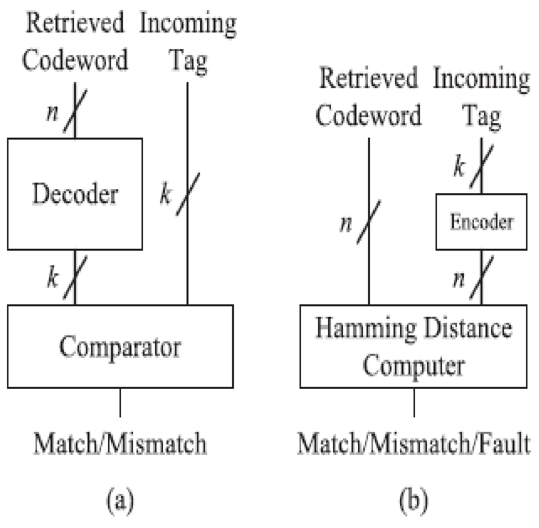


Fig. 2. (a) Decode-and-compare architecture and (b) encode-and-compare architecture.

The comparison is to examine how many bits the two codewords differ, not to check if the two codewords are exactly equal to 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 $0 < d \leq t_{\max}$, X will match Y provided at most $t_{\max}$ errors in Y are corrected.

3) If $t_{\max} < d \leq r_{\max}$, 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 $r_{\max} < d$, 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 dis-tance, [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 codewords. 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:

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

The final accumulated value indicates the range of d. As the compul-sory 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

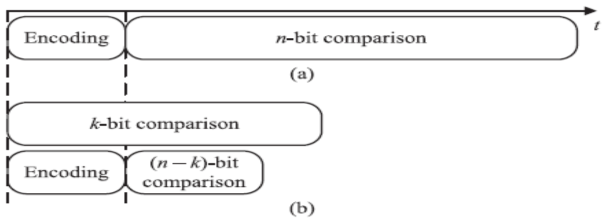


Fig. 3. Timing diagram of the tag match in (a) direct compare method [6] and (b) proposed architecture

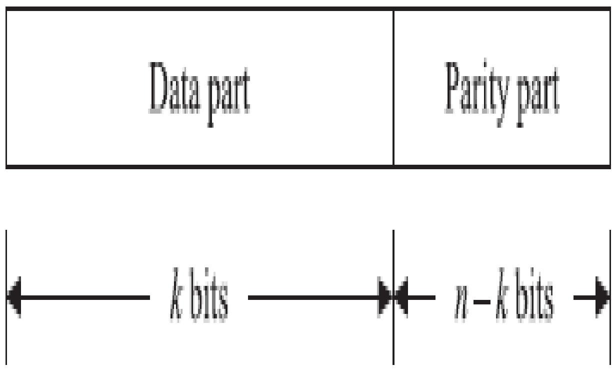


Fig. 4. Systematic representation of an ECC codeword.

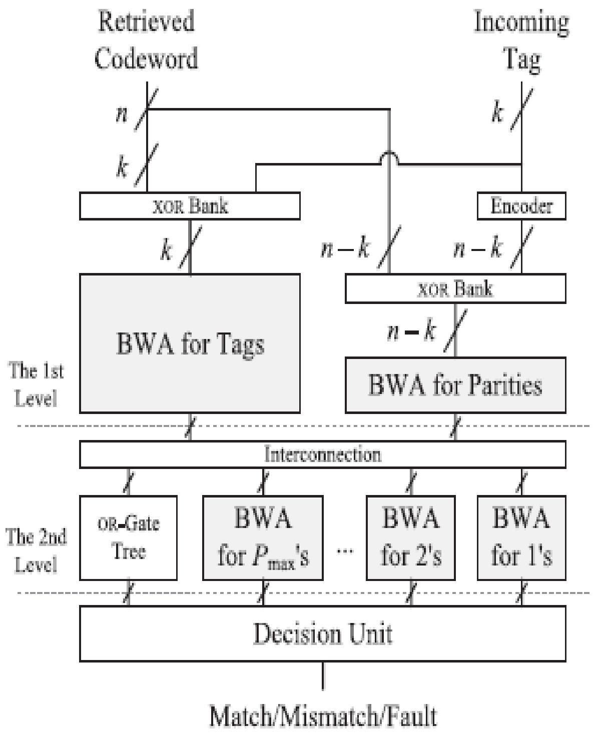


Fig. 5. Proposed architecture optimized for systematic codewords.

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 codewords 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 codeword 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 codeword is exactly the same as the incom-ing 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 .

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.

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 one’s 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.

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 accumu-lators (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 multiple stages of HAs as shown in Fig. 6(a), where each output bit of a HA

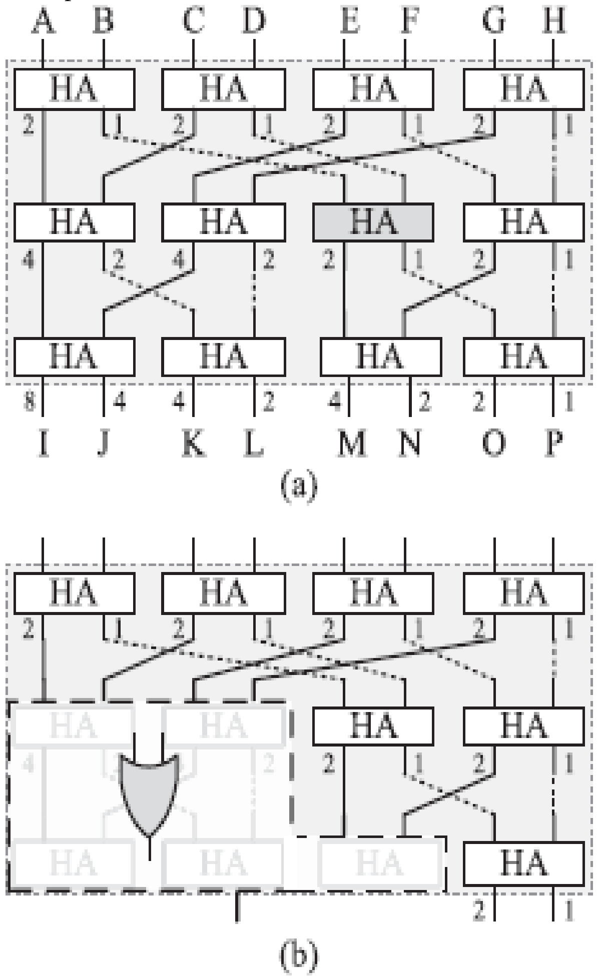


Fig. 6. Proposed BWA. (a) General structure and (b) new structure revised for the matching of ECC-protected data. Note that sum-bit lines are dotted for visibility.

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

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

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 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

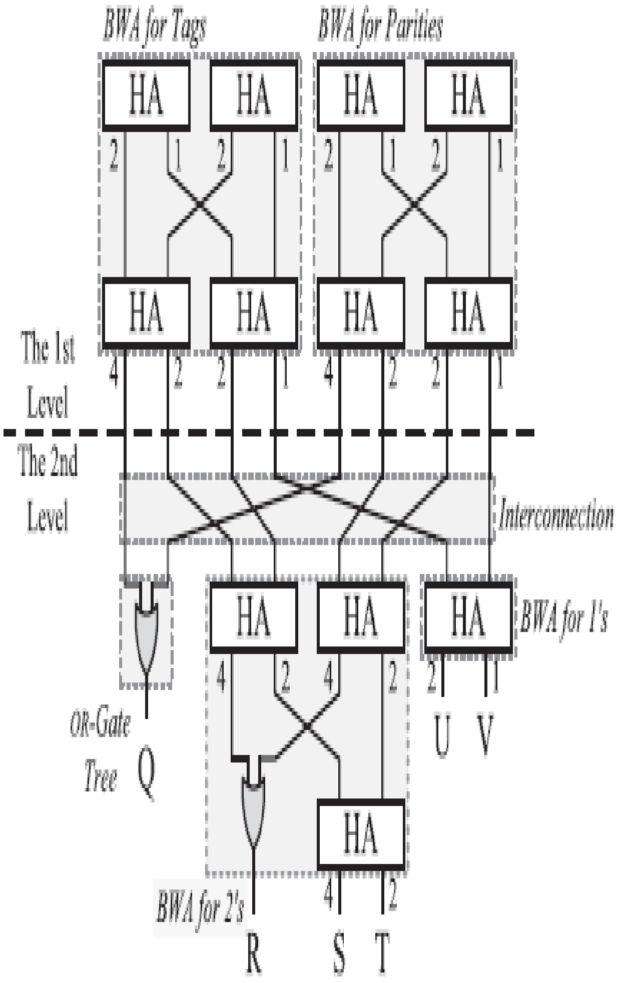


Fig. 7. First and second level circuits for a (8, 4) code

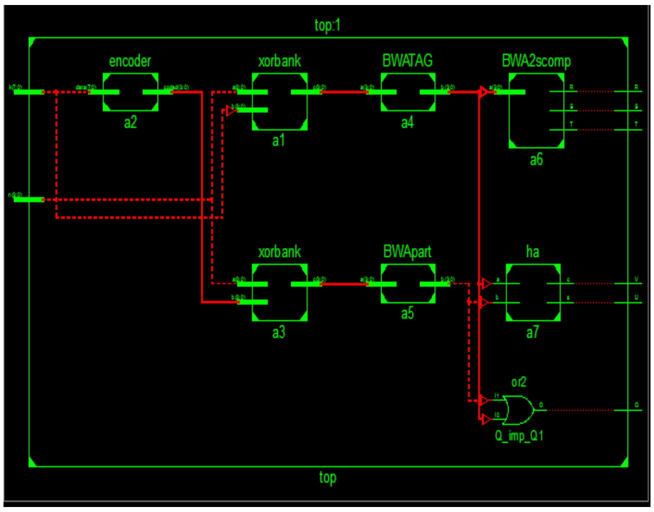
TABLE I
Truth Table Of The Decision Unit F Or A (8, 4) CODE

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

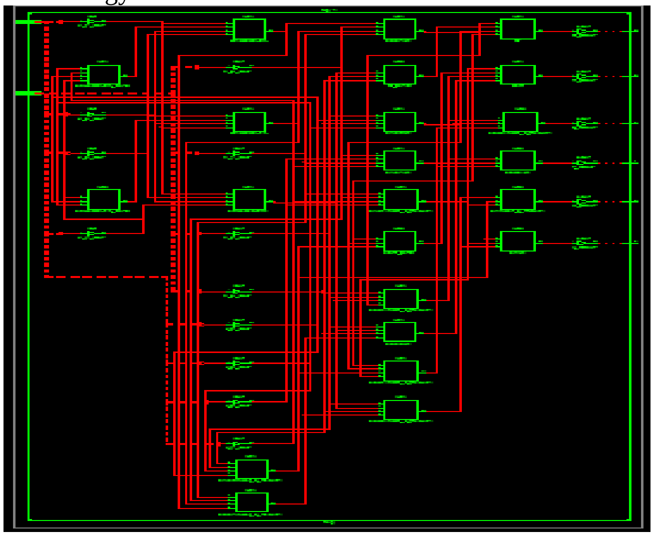
The corresponding first and second level circuits are shown in Fig. 7. Note that the encoder and XOR banks are not drawn in Fig. 7 for the sake of simplicity. Since $r_{\max} = 2$, $P_{\max} = 2$ and there are only two BWAs dealing with weights 2 and 1 at the second level. As the bits of weight 4 fall 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.

IV. RESULTS

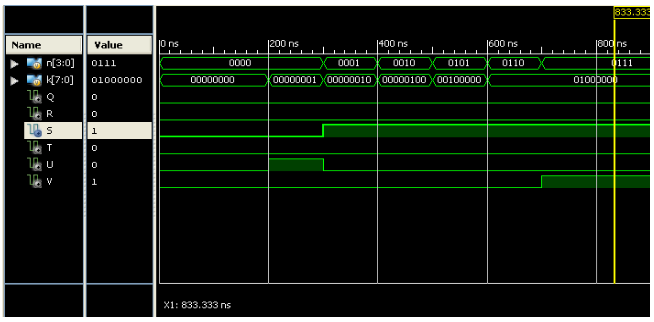
RTL Schematic:



TechnologySchematic:



Simulated Waveform:



V.CONCLUSION

To reduce the latency and hardware complexity, a new architecture has been presented for matching the data protected with an ECC using multiplexer based encoding. 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 technique. The power consumption is drastically reduced from 0.25274mw to 0.1875mw with a slight increase in latency. The functionality is verified using ISE simulator and the synthesis is carried out using XILINX synthesizer by developing the RTL using VERILOG HDL.

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 EncodedWith 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 zEnterprise 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.
[6] 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
[7] W. Wu, D. Somasekhar, and S.-L. Lu, "Direct compare of information coded with error-correcting codes," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 11, pp. 2147–2151,Nov. 2012.
[8] S. Lin and D. J. Costello, Error Control Coding: Fundamentals and Applications, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.