Volume 2, Issue 12, December-2015, pp. 849-855 ISSN (O): 2349-7084

International Journal of Computer Engineering In Research Trends

Available online at: www.ijcert.org

A Novel Architecture for Matching the Data Protected With an Error-Correcting Code (ECC)

[1]Sk.Parvin Bhano, [2]T.Vineela

[1](M.Tech) –VLSI & ES, Dept of ECE, Vasireddy Venkatadri Institute of Technology (VVIT),
[2]Assistant Professor, Dept of ECE, Vasireddy Venkatadri Institute of Technology (VVIT), Nambur (V), Guntur(Dt),Andhra Pradesh, India.

Abstract:- In this paper we presented a novel architecture for matching the data protected with an Error-Correcting Code (ECC) which proposed to reduce latency and complexity where Data comparison is widely used in computing system to perform so many operations. Where incoming information is needs to be compared with a piece of stored data to locate the matching entry. If both incoming bits and stored bits are matching means there is no error if mismatched means some type of error will occur like random error or burst error. To detect and correct the error here error correcting codes are used. 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 and also demonstrates LDPC coding and decoding for Error Correcting Codes further more examines whether the incoming data matches the stored data if a certain number of burst errors are corrected.

Index Terms— Low Power and Low Complexity in Chip, Systematic Error Correcting Code.

I. 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.

II. DATA COMPARISION METHODS WITH EXISTING SYSTEM

II. DATA COMPARISION METHODS WITH EXISTING SYSTEM

2.1 Direct Compare Method

This describes the conventional decode-and-compare architecture. Let us consider a cache memory where a kbit 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, 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

Direct compare method is one of the most recent solutions for the matching problem. The direct compare method 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.

2.3 Sa-Based Approach

SA-based approach is the one where a special counter is constructed with an additional` building block called saturating adder (SA).The SA-based direct compare architecture reduces the latency and hardware complexity by resolving the aforementioned drawbacks

III.ADVANCED DATA COMPARISION METHODS WITH PROPOSED SYSTEM

Existing 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.

3.1. XOR Bank

XOR bank represents the array of bit-wise comparators (exclusive OR gates). It 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 codeword's. The output from the XOR bank is then fed into BWA consisting of half adders (HAs). The numbers of 1's are accumulated by passing the value through the BWA.

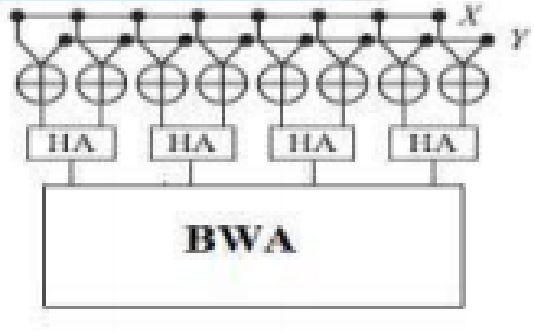


Fig 1. XOR bank structure

3.2. Butterfly Formed Weight Accumulator

The proposed architecture grounded on the data path design is given below. 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

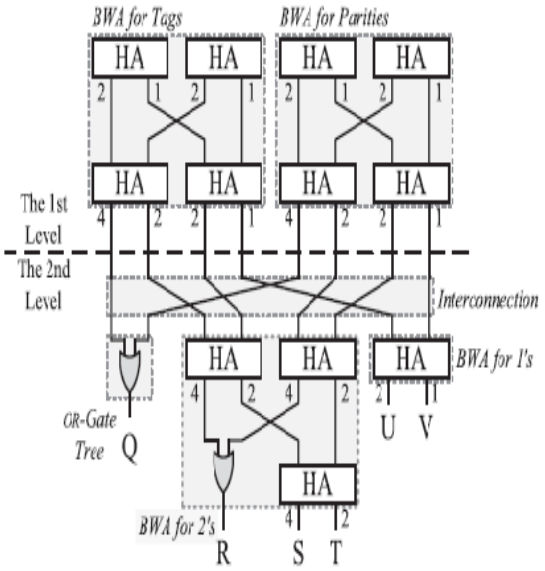


Fig 3. BWA Architecture.

IV.SYSTEM STUDY ON PROPOSED WORK

A binary parity check code is a block code: i.e., a collection of binary vectors of fixed length n. The symbols in the code satisfy r parity check equations of the form: $x_a \oplus x_b \oplus x_c \oplus \ldots \oplus x_z = 0$ where $\oplus$ means modulo 2 additions and $x_a, x_b, x_c, \ldots, x_z$ are the code symbols in the equation.

4.1. LDPC Code:

The percentage of 1's in the parity check matrix for a LDPC code is low. A regular LDPC code has the property that: – every code digit is contained in the same number of equations, – each equation contains the same number of code symbols. An irregular LDPC code relaxes these conditions.

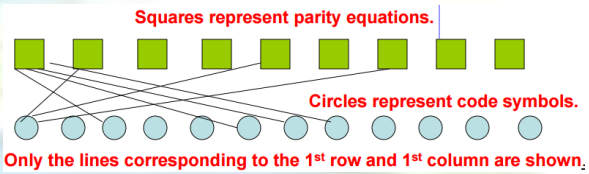
The Equations for A Simple LDPC Code with n=12

$$c_3 \oplus c_6 \oplus c_7 \oplus c_8 = 0$$
$$c_1 \oplus c_2 \oplus c_5 \oplus c_{12} = 0$$
$$c_4 \oplus c_9 \oplus c_{10} \oplus c_{11} = 0$$
$$c_2 \oplus c_6 \oplus c_7 \oplus c_{10} = 0$$
$$c_1 \oplus c_3 \oplus c_8 \oplus c_{11} = 0$$
$$c_4 \oplus c_5 \oplus c_9 \oplus c_{12} = 0$$
$$c_1 \oplus c_4 \oplus c_5 \oplus c_7 = 0$$
$$c_6 \oplus c_8 \oplus c_{11} \oplus c_{12} = 0$$
$$c_2 \oplus c_3 \oplus c_9 \oplus c_{10} = 0.$$

The Parity Check Matrix for the Simple LDPC Code

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
0	0	1	0	0	1	1	1	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	1	1	1	0
0	1	0	0	0	1	1	0	0	1	0	0
1	0	1	0	0	0	0	1	0	0	1	0
0	0	0	1	1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0	1	1	0	0

The Graph for the Simple LDPC Code



DMC Encoding

Because of high-speed caches and main memories, which are prone to soft errors, error correcting codes are used in the design and, more recently, in the design of on chip memories. For the encoding Decimal matrix code (DMC) is proposed to assure reliability in the presence of MCUs with reduced performance overheads, and a 4-bit word is encoded based on the proposed technique.

First, during the encoding process, information bits i are fed to the DMC encoder, and then the horizontal redundant bits H and vertical redundant bits V are obtained from the DMC encoder. When the encoding process is completed, the obtained DMC codeword is stored in the memory. Second, the horizontal redundant bits H are produced by performing xor operation of selected symbols per row. Third, the vertical redundant bits V are obtained by xor operation among the bits per column. It should be noted that both divide-symbol and arrange matrix are implemented in logical instead of in physical. Therefore, the proposed DMC does not require changing the physical structure of the memory.

i1	i0	H0
i3	i2	H1
V1	V0	

Fig 1. 4-bit DMC logical organization

The 4-bit word has been divided into two symbols of 2-bit.k1 = 2 and k2 = 2 have been chosen simultaneously.H0 and H1 are horizontal check bits;V0 and V1 are vertical check bits.

Example LDPC Encoder

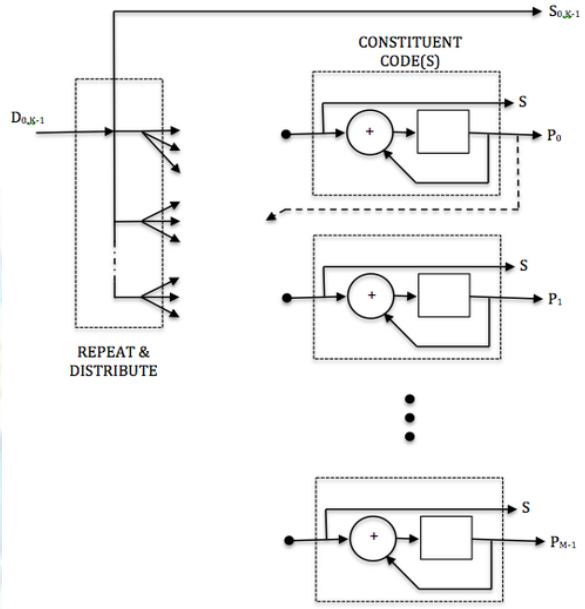
During the encoding of a frame, the input data bits (D) are repeated and distributed to a set of constituent encoders. The constituent encoders are typically accumulators and each accumulator is used to generate a parity symbol. A single copy of the original data (S0,K-1) is transmitted with the parity bits (P) to make up the code symbols. The S bits from each constituent encoder are discarded.

The parity bit may be used within another constituent code.

In an example using the DVB-S2 rate 2/3 code the encoded block size is 64800 symbols (N=64800) with 43200 data bits (K=43200) and 21600 parity bits (M=21600). Each constituent code (check node) encodes 16 data bits except for the first parity bit which encodes 8 data bits. The first 4680 data bits are repeated 13 times (used in 13 parity codes), while the remaining data bits are used in 3 parity codes (irregular LDPC code).

For comparison, classic turbo codes typically use two constituent codes configured in parallel, each of which encodes the entire input block (K) of data bits. These constituent encoders are recursive convolutional codes (RSC) of moderate depth (8 or 16 states) that are separated by a code interleaver which interleaves one copy of the frame.The LDPC code, in contrast, uses many low depth constituent codes (accumulators) in parallel, each of which encode only a small portion of the input frame. The many constituent codes can be viewed as many low depth (2 state) 'convolutional codes' that are connected via the repeat and distribute operations. The repeat and distribute operations perform the function of the interleaver in the turbo code.

The ability to more precisely manage the connections of the various constituent codes and the level of redundancy for each input bit give more flexibility in the design of LDPC codes, which can lead to better performance than turbo codes in some instances. Turbo codes still seem to perform better than LDPCs at low code rates, or at least the design of well performing low rate codes is easier for Turbo Codes.As a practical matter, the hardware that forms the accumulators is reused during the encoding process. That is, once a first set of parity bits are generated and the parity bits stored, the same accumulator hardware is used to generate a next set of parity bits.



Encoding LDPC codes is roughly done like that: Choose certain variable nodes to place the message bits on. And in the second step calculate the missing values of the other nodes. An obvious solution for that would be to solve the parity check equations. This would contain operations involving the whole parity-check matrix and the complexity would be again quadratic in the block length. In practice however, more clever methods are used to ensure that encoding can be done in much shorter time. Those methods can use again the spareness of the parity-check matrix or dictate a certain structure3 for the Tanner graph.

Decoding of LDPC Codes by Message Passing on the Graph

Decoding is accomplished by passing messages along the lines of the graph. The messages on the lines that connect to the i-th bit node, ci , are estimates of Pr[ci =1] (or some equivalent information). At the nodes the various estimates are combined in a particular way.Each bit node is furnished an initial estimate of the probability it is a 1 from the soft output of the channel. The bit node broadcasts this initial estimate to the parity nodes on the lines connected to that bit node.But each parity node must make new estimates for the bits involved

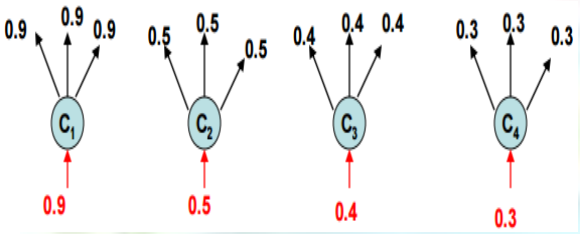
in that parity equation and send these new estimates (on the lines) back to the bit nodes.

Decoding of Simple Example

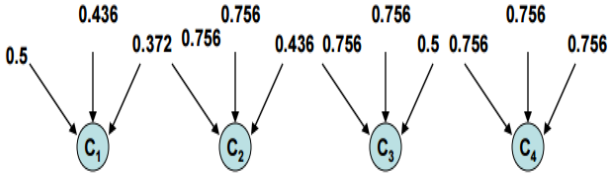
Suppose the following Pr[Ci=1], i=1, 2, …, 12 are obtained from channel: 0.9 0.5 0.4 0.3 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 • We now watch the decoder decode.

Decoding of Simple Example: First 4 Bit Nodes Only

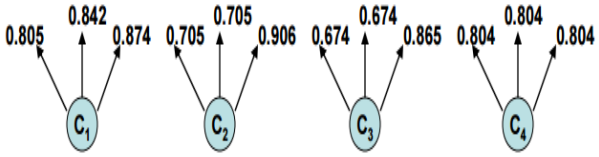
Initial broadcast from first 4 bit nodes:



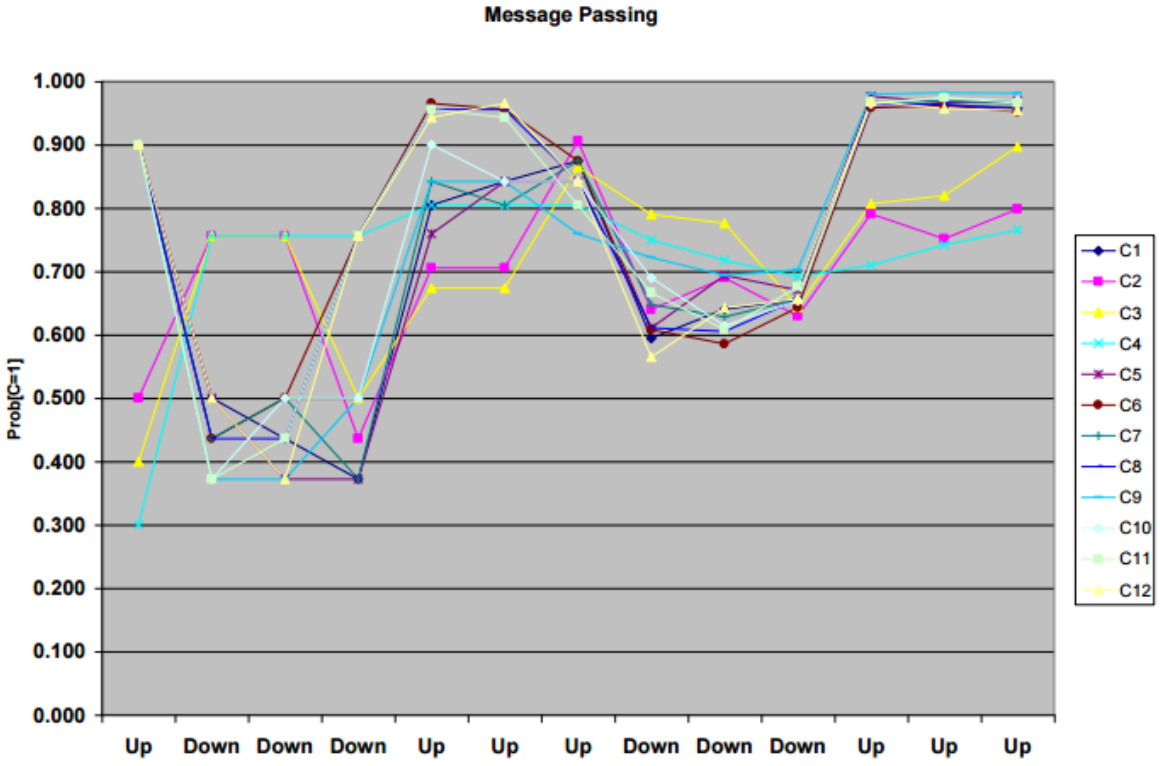
Transmission from parity nodes to these 4 bit nodes:



Next transmission from the first 4 bit nodes:



Messages Passed To and From All 12 Bit Nodes



V. SIMULATION RESULTS

Encoder result

The Encoder gives its output by encoding the given input message with check bits. Finally the output of encoder be code-word the combination of message and check bits.

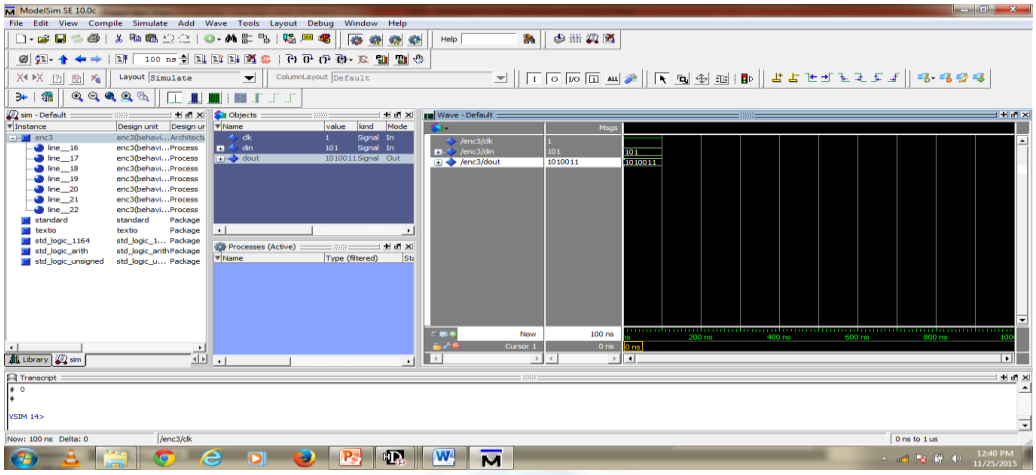


Fig 4. Encoder simulation result.

Decoder result

The Decoder takes the output of Encoder, and gives the exact input of user, without any noise and disturbance

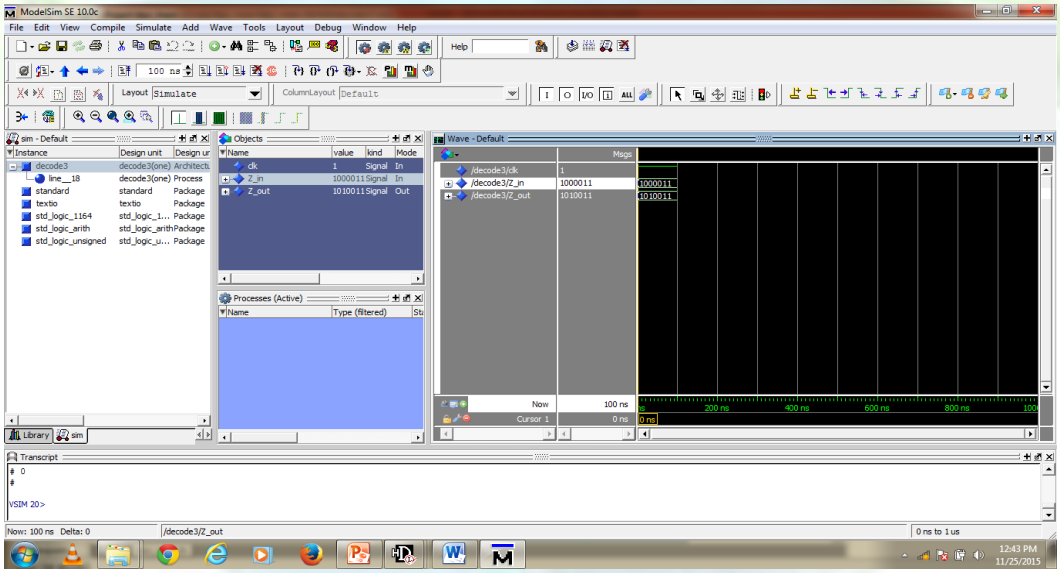


Fig 5. Decoder simulation result.

Encoder/decoder result

Coding for this encoding and decoding is done in VHDL using HDL designer Series and simulation results are reported below Here I am giving initial input as 101 and its encoded codeword (d) is 1010011.Then to the encoded 7 bit codeword(d) noise of 7 bit "0110000" is added to model the effect of noisy channel. Due the added noise some bits in the codeword gets flipped and the resulting corrupted codeword i.e total encoder output is (e)1100011. by bit flipping decoding error bit is flipped and the original message vector is decoded i.e; The total encoder output is (e)1100011 will be given as input to decoder.Then the decoder decodes the encoder output(e) and the output dout as 1010111

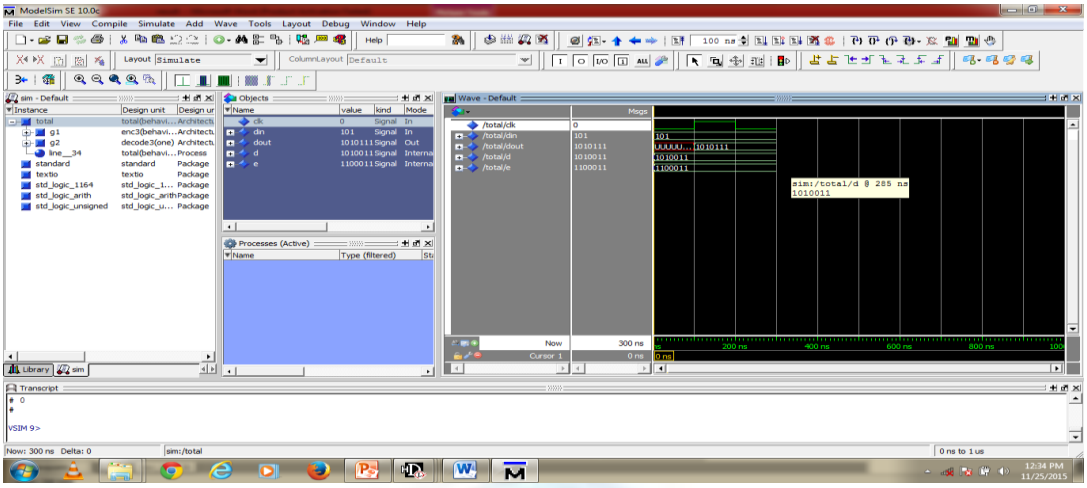


Fig 6. Encoder/Decoder simulation result.

V.CONCLUSSION

In this method, we make use of the DMC a system to guarantee the reliability of memory. The proposed insurance code uses a decimal calculation to identify blunders so that more mistakes were identified and revised. To lessen the equipment multifaceted nature and inactivity, another building design has been introduced for coordinating the information ensured with an ECC.To diminishes the idleness; the examination of the information is parallelized with the encoding process that produces the equality data. The inherent parallelism in decoding LDPC codes suggests their use in high data rate systems LDPC codes are well worthwhile investigating.

REFERENCES

[1] Byeong Yong Kong, Jihyuck Jo, Hyewon Jeong, Mina Hwang,Soyoung Cha, Bongjin Kim, and InCheol Park "Low-Complexity Low-Latency Architecture for Matching Of Data EncodedWith Hard Systematic ErrorCorrecting Codes" IEEE transactions on (vlsi) systems, vol. 22, no. 7, July 2014

[2] J. D. Warnock, Y.-H. Chan, S. M. Carey, H. Wen, P. J. Meaney, G. Gerwig and W. V. Huott "Circuit and physical design implementation of the microprocessor chip for the Enterprise system," IEEE J. Solid-State Circuits, vol. 47, no. 1, pp. 151–163, Jan. 2012. [3].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.

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

[5] Y. Lee, H. Yoo, and I.-C. Park, "6.4Gb/s multithreaded BCH encoder and decoder for multi-channel SSD controllers," in ISSCC Dig. Tech. Papers, 2012, pp. 426–427 .

[6] M. Tremblay and S. Chaudhry, "A thirdgeneration 65nm 16-core 32-thread plus 32-scoutthread CMT SPARC processor," in ISSCC. Dig. Tech. Papers, Feb. 2008, pp. 82–83.

[7] 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

