ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

VLSI Implementation of Soft Bit Flipping Decoder for Geometric LDPC Codes

A.Sushilkumar Singh, K. Jyothi

P.G.Student (M.Tech) in VLSI and ES, GIET, Rajahmundry, A.P., India

Associate Professor, Department of ECE, GIET, Rajahmundry, A.P., India

ABSTRACT: As the Technology improves digital systems are become more prominent and the reliability and security of memories are essential considerations. As technology scales, memory devices become larger and more eminent error correction codes are required to protect memories from soft errors. Low Density Parity Check (LDPC) Codes are the class of linear block codes which provide large collection of data transmission channels while simultaneously feasible for implementable decoders.One specific type of LDPC codes, namely EG- LDPC are used due to their fault secure detection capability, higher reliability and lower area overhead.In this paper, a low-complexity high-performance algorithm is introduced for decoding of LDPC codes.The developed soft-bit-flipping (SBF) algorithm having advantages of bit-flipping (BF) algorithm and reliability of improve error performance. A hybrid decoding scheme comprised of the BF and SBF algorithms is also proposed to shorten the decoding time.

KEY TERMS: LDPC Codes, soft-bit-flipping (SBF) algorithm, Bit-Flipping (BF) Algorithm**.**

I. INTRODUCTION

Now a day, data communication has become essential part of life and a lot of data is being transferred. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. There also are different ways of hacking, where the intruder modifies the data while communication. So to protect the confidentiality of the data and to retain the correctness of the data, secure communication is very important.

There are different methods of implementing the secure communication. Each method has its own advantages and disadvantages. This project is an improvement on most of the existing methods for secure communication.

A)LDPC Codes

Low-density parity-check (LDPC) codes are a class of linear block codes. The name comes from the characteristic of their parity-check matrix which contains only a few 1's in comparison to the amount of0's. Their main advantage is that they provide a performance which is very close to the capacity for a lot of different channels and linear time complex algorithms for decoding. Furthermore are they suited for implementations that make heavy use of parallelism. They were first introduced by Gallager in his PhD thesis in 1960. But due to the computational effort in implementing coder and encoder for such codes and the introduction of Reed-Solomon codes, they were mostly ignored until about ten years ago.

B) Fundamentals of Linear Block Codes

- The structure of a code is completely described by the generator matrix G or the parity check matrix H.
- The capacity of correcting symbol errors in a codeword is determined by the minimum distance (d_{min}).
  i. d_{min} is the least weight of the rows in G.
  ii. d_{min} is the least number of columns in H that sum up to 0.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \text{_____} 1$$

Fig1:LDPC matrix

The Sum Product Algorithm is used to find the error performance but this algorithm is very complex to find the error. And also this algorithm occupies much area and decoding time. So to overcome this problem we can use Bit Flipping Algorithm. It is very easy to find out the error with low complexity and less area. The parallel and pipeline architectures are used to improve the through put.

Traditionally, LDPC codes have been defined pseudo-randomly. This has proven very effective at producing good decoding performances, particularly for very long codes. However, a random construction method also means that code properties are not guaranteed for any individual code, and are thus not easy to control or even determine. As well, implementation complexity is increased by the need to store, and encodes, codes described by random parity-check matrices.

In particular, their potentially high encoding complexity, which is in general quadratic in the code length, is a serious shortcoming of LDPC codes and compares poorly with the linear time encoding of turbo codes. Finding computationally efficient encoders is therefore important for LDPC codes to be considered as serious contenders for replacing turbo codes in future generations of forward error correction devices. Several approaches have been suggested, including the manipulation of the parity-check matrix to establish that while the complexity is, strictly speaking, quadratic, the actual number of encoding operations grows essentially linearly with code length. The work of Lucas et al. and Kou et al. demonstrates that as well as providing implementation advantages the algebraic finite geometry codes can significantly outperform equivalent length and rate randomly constructed codes when both are decoded with the sum-product algorithm. The excellent performance of the finite geometry codes was the motivation for this work, by showing that codes which perform excellently with sum-product decoding can be constructed without the need for random interleavers or random parity-check matrices.

With this background in mind the problem considered in this thesis is the design of algebraic error correction codes for use with sum-product decoding or, more succinctly, the design of LDPC codes. The central idea of this thesis is to apply combinatorial theory to the design of new algebraic LDPC codes. The potential benefits of considering codes from combinatorial designs are guaranteed code properties, parameter flexibility, and ease of implementation including reduced storage requirements and reduced encoding complexity.

II. REVIEW OF LITERATURE

A) SUM PRODUCTALGORITHM

The sum-product algorithm is an iterative algorithm for computing either exact marginal's (on trees), or approximate marginal's (for graphs with cycles). It operates in a distributed manner, with nodes in the graph exchanging statistical information via a sequence of "message-passing" updates. For tree-structured graphical models, the updates can be derived as a form of non-serial dynamic programming, and are guaranteed to converge and compute the correct marginal distributions at each node. However, the updates are routinely applied to more general graphs with cycles, which is the application of interest in this paper. Here we describe the more general family of reweighted sum-product algorithms, which include the ordinary sum-product updates as a particular case.

Bayesian networks are so named because they can be solved using Bayes rule:

$$P(A|B) = \frac{P(A|B)P(A)}{P(B)}$$

Unfortunately, for large networks, Bayes' simple rule becomes unruly, and so it is necessary to use a message-passing algorithm. The sum-product algorithm is a general form of the forward-backward algorithm. It attempts to

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

compute various marginal functions associated with the global function in a factor graph. Once one or more nodes have been observed to be of some value, messages are passed inward from an arbitrary set of nodes at the edge of a graph. When they reach an edge, their direction is reversed, and when they reach their origin, they are absorbed.

Message Passing

The first messages are passed from some set of singly-connected function nodes to the variable nodes that they depend upon. No computation is necessary in this step because the messages are simple identity messages that specify the apriority knowledge stored in the function that creates them. The variable nodes then pass the same identity message along to the next function node that they are connected to. These steps are marked ``1" and ``2" respectively in below figure 2.

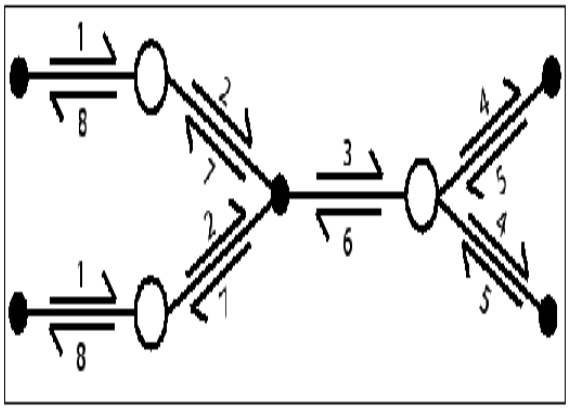


Fig 2: Sum Product Message Passing Diagram

Assuming the neighbouring function node is connected to more than one variable; it must wait for all but one of its neighbours to send it a message. Once those messages are received, it creates a message of its own and passes that message along to the variable that did not send it a message. This message is created from the messages passed to it and its internal apriori knowledge. Equation 2 shows how this message is created.

$$x = n(f) \sum_{\sim\{x\}} \left(f(x) \prod_{y \in n(f)\setminus\{x\}} \mu_{h \to x}(y) \right) \quad \text{——————} \quad 2$$

In equation 2, x=n(f) is the set of arguments to the function f, and $\mu_{h \to x}(y)$ is a message from some neighbour y to the function. This marginalization is somewhat difficult to understand, but is quick to compute. In English, for each possible argument to f, we compute the value of the f, which is a piece of apriority knowledge, then multiply it by each message we have received according to that same argument. The sum of these computations becomes the message to the next variable, marked ``3" in figure 2.

Step 4 is the same as step 2. In step 5, the functions at the periphery of the graph reverse the flow of messages, performing the same marginalization as in step 3. Step 6 requires a different type of marginalization:

$$\prod_{h \subset n(x)\setminus\{f\}} \mu_{h \to x}(x) \quad \text{——————} \quad 3$$

In this case, x is the variable creating the message, and n(x)\{f} is the set of functions that have sent x a message. n(x)\{f} does not include the message received while the messages were travelling in the other direction. Functions that receive only a single message may omit this marginalization, as it is trivial, and simply pass the message on to its one neighbour not in n(x)\{f} ,g threshold Ĕ's.

Iterative decoding of binary low-density parity-check (LDPC) codes using the sum-product algorithm (SPA) has recently been shown to approach the capacity of the additive white Gaussian noise (AWGN) channel within 0.0045 dB.Efficient hardware implementation of the SPA has become a topic of increasing interest. The direct implementation of the original form of SPA has been shown to be sensitive to quantization effects. In addition, using likelihood ratios can substantially reduce the required quantization levels.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

An implication of the SPA that reduces the complexity of the parity check update at the cost of some loss in performance. This implication has been derived by operating in the log-likelihood domain. Recently, a new reduced complexity decoding algorithm that also operates entirely in the log-likelihood domain. It bridges the gap in performance between the optimal SPA and finally, low complexity software and hardware implementations of an iterative decoder for LDPC codes suitable for multiple access application.

Here we present efficient implementations of the SPA and describe new reduced-complexity derivatives thereof. In our approach, log-likelihood ratios (LLR) are used as messages between symbol and parity-check nodes. It is known that in practical systems, using LLRs offers implementation advantages over using probabilities or likelihood ratios, because multiplications are replaced by additions and the normalization step is eliminated. The family of LDPC decoding algorithms presented here is called LLR-SPA. The unified treatment of decoding techniques for LDPC codes presented here provides flexibility in selecting the appropriate design point in high-speed applications from a performance, latency, and computational complexity perspective. In particular, serial and parallel implementations are investigated, leading to trellis and tree topologies, respectively. In both cases, specific core operations similar to the special operations defined in the log-likelihood algebra of are used.

This formulation not only leads to reduce complexity LDPC decoding algorithms that can be implemented with simple comparators and adders but also provides the ability to compensate the loss in performance by using simple look-up tables or constant correction terms.

III.PROPOSED METHOD

A) Block Diagram

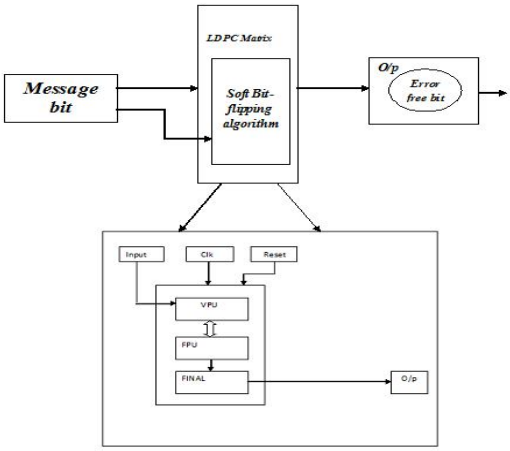


Fig 3: block diagram

B) BIT-FLIPPING ALGORITHM

Assume that we have a binary linear code defined using a specific Code graph(or Tannergraph). A Code graph is a bipartite graph with the variable nodes on one side and checknodes on the other side. As we saw in the previous lectures, each check node imposes aparity constraint on the set of variable nodes connected to it. In the previous lecture, wedefined a decoding algorithm which we claimed has a guarantee of success for all Codegraphs which are expander and have fixed variable node degrees. Before making thesedefinitions more precise, we review the definition of an expander graph:

Definition 1: A factor graph is a (ϵ, δ)-expander if for every subset U of the set of variable nodes of size bounded by $|U| \leq \epsilon N$, it holds that $|\partial U| > \delta|U|$, where ∂U is the set of factornodes adjacent to at least one node in U.

Assume that we have transmitted some codeword W = (W1,...,Wm) and the decoderhas received the output sequence Y = (Y1,..., Ym) over a BSC channel with parameter p(meaning P (Yi = Xi) = 1 − p).

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

The bit flipping algorithm starts out with sequence Y = (Y1,...,Ym). Eachchecknodeintroduces a constraint. As we saw in the previous lecture, in each step, the algorithm tries to find a bit, flipping of which reduces the number of unsatisfied clauses. The algorithmstops when for every bit, the number of unsatisfied constraints is less or equal to the numberof satisfied constraints. Since the number of unsatisfied constraints reduces in each step,the algorithm has to stop. We will prove the following theorem:

Theorem 1: Consider a Code graph where all variable nodes have degree l, and the graphs (∈, δ)-expander. Then the bit- flipping algorithm will correct any pattern of at most$N \in /2$errors.

Proof: Without loss of generality we can assume the all zero codeword has been sent(since the code is linear). Thus the hamming weight of the received codeword equals to thenumber of bits we have received in error. Let

- X(t) denote the string at the t^{th}iteration of the algorithm. Since we start out withthe received message, we have X(0) = Y .
- W(t) denote the hamming weight of X(t).
- u(t) denote the number of unsatisfied check nodes.
- s(t) denote the number of satisfied checks that are adjacent to at least one Xi suchthat Xi = 1.

Since the number of errors is at most$N \in /2$, we have w(0) ≤$N \in /2$.It is clear from the bit-flipping algorithm that u(t) is a decreasing function of t. Thefollowing two propositions thus imply the result stated at Theorem 1.

Proposition 1: Let t^*be the iteration in which the algorithm stops, then w(t*) is eitherzero, or is greater than or equal to N∈.

Proof: Assume that 0 < w (t*) < N∈. Consider the nodes which are one. Because ofexpansion, the number of check nodes adjacent to those nodes that are one is at least $\frac{3}{4}$lw(t*). Let set T denote these set of check nodes.

The check nodes adjacent to variable nodes that are ones are either counted in u(t) or s(t). Therefore |T | = u(t*) + s(t*) >$\frac{3}{4}$lw(t^*).On the other hand, since every satisfied check is adjacent to at least two variable nodes with Xi = 1, the number of edges going out from the set T is at least u(t^*) + 2s(t*). But since the degree of each variable node is l, the number of edges going out from set T is exactly equal to lw(t*), and therefore we get u(t*) + 2s(t*) ≤ lw(t*). This inequality together with u(t∗) + s(t∗) >$\frac{3}{4}$lw(t∗), one can easily show, imply that u(t*) >$\frac{l}{2}$lw(t*).

But u(t*) >$\frac{1}{2}$lw(t*) implies that there exist at least one variable node with Xi = 1 thatis adjacent to more than$\frac{l}{2}$check nodes. This is a contradiction since this variable nodeis adjacent to more unsatisfied nodes than satisfied nodes and thus the algorithm couldn'thave stopped at iteration t*.

Proposition 2: The algorithm cannot terminate with a string of weight greater than N∈.

Proof: It is enough to prove that there is no t such that w(t) = N∈. Assume otherwise thatw(t) = N∈. The above inequalities show that u(t) >$\frac{l}{2}$w(t) =$\frac{l}{2}$N∈. On the other hand,u(t) ≤ u(0) =$\frac{lN\epsilon}{2}$which is a contradiction.

C) BF Decoding

BF decoding of LDPC codes was devised by Gallager in the early 1960's [1,2]. When detectable errors occur during the transmission, there will be parity failures in the syndrome $s = s_1, s_2, \ldots . sj$and some of the syndrome bits are equal to 1. BF decoding is based on the change of the number of parity failures in $z.hj: 1 \leq j \leq J$when a bit in the received sequence **z** is changed.

First the decoder computes all the parity check sums based on (24) and then changes any bit in the received vector z that is contained in more than some fixed number$\boldsymbol{\delta}$ of unsatisfied parity check equations. Using these new values, the parity check sums are recomputed, and the process is repeated until the parity check equations are all satisfied. This decoding is an iterative decoding algorithm. The parameter$\boldsymbol{\delta}$, called **threshold**, is a design parameter which should be chosen to optimize the error performance while minimizing the number of computations of parity check sums. The value of$\boldsymbol{\delta}$ depends on the code parametersρ, γ, dmin(C) and the signal-to-noise ratio (SNR).

If decoding fails for a given value of Æ , then the value of Æ can be reduced to allow further decoding iterations. For error patterns with number of errors less than or equal to the error correcting capability of the code, the decoding will be completed in one or a few iterations. Otherwise, more decoding iterations are needed. Therefore, the number of decoding iterations is a random variable and is a function of the channel SNR. A limit may be set on the number of iterations. When this limit is reached, the decoding process is terminated to avoid excessive computations.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

Due to the nature of low density parity checks, the above decoding algorithm corrects many error patterns with number of errors exceeding the error correcting capability of the code.

A very simple BF decoding algorithm is given below:

Step 1: Compute the parity check sums (syndrome bits). If all the parity check equations are satisfied (i.e., all the syndrome bits are zero), stop the decoding.

Step 2: Find the number of unsatisfied parity check equations for each code bit position, denoted f_i, i=0,1,....,n-1.

Step 3: Identify the set Ω of bits for which f_i i is the largest.

Step 4: Flip the bits in set Ω

Step 5: Repeat steps 1 to 4 until all the parity check equations are satisfied (for this case, we stop theiteration in step 1) or a predefined maximum number of iterations is reached.

BF decoding requires only logical operations. The number of logical operations N_{BF}performed for each decoding iteration is linearly proportional to Jρ (or ηγ), say $N_{BF} = K_{BF}$,Jρ where the K_{BF} constant KBF depends on the implementation of the BF decoding algorithm. Typically, K_{BF} is less than three. The simple BF decoding algorithm can be improved by using adaptive thresholds δ Of course; this improvement is achieved at the expense of more computations. EG- and PG-LDPC codes perform well with the BF decoding due to the large number of check sums orthogonal on each code bit.

IV.SIMULATION RESULTS

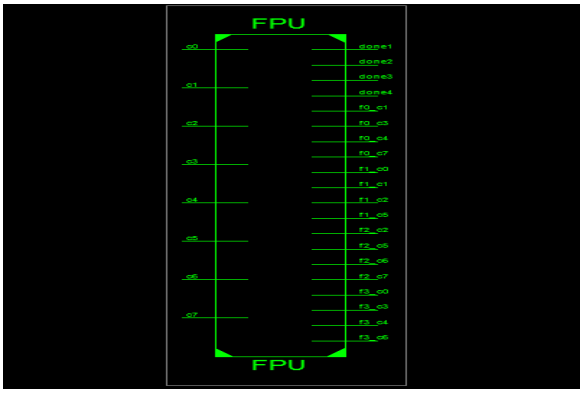
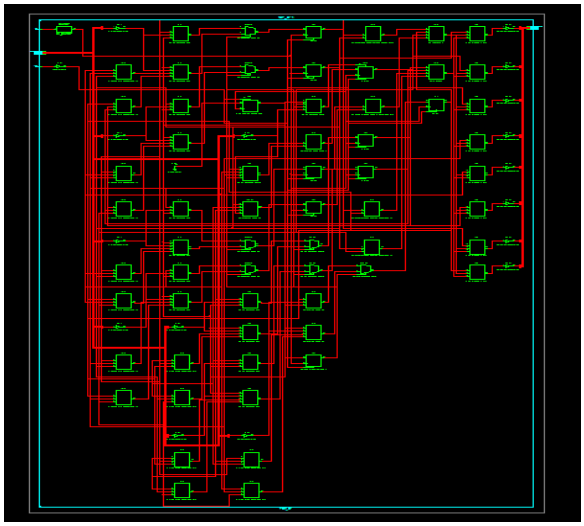


Fig 4: Block diagram



ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

Fig 5: Technology schematic

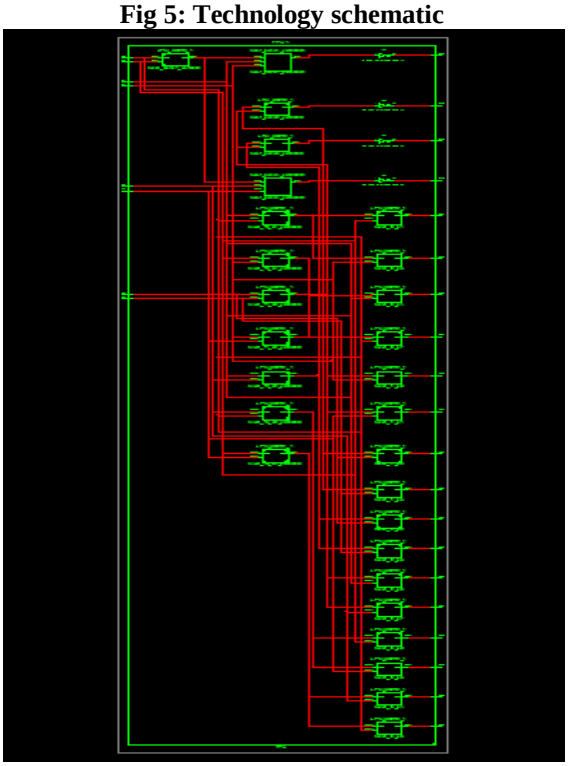


Fig 6: RTL schematic

Table I: Device Utilization Summary

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	10	7,168	1%	
Number of 4 input LUTs	46	7,168	1%	
Number of occupied Slices	24	3,584	1%	
Number of Slices containing only related logic	24	24	100%	
Number of Slices containing unrelated logic	0	24	0%	
Total Number of 4 input LUTs	46	7,168	1%	
Number of bonded IOBs	18	141	12%	
Number of BUFGMUXs	1	8	12%	
Average Fan-out of Non-Clock Nets	3.44			

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 11, November 2014

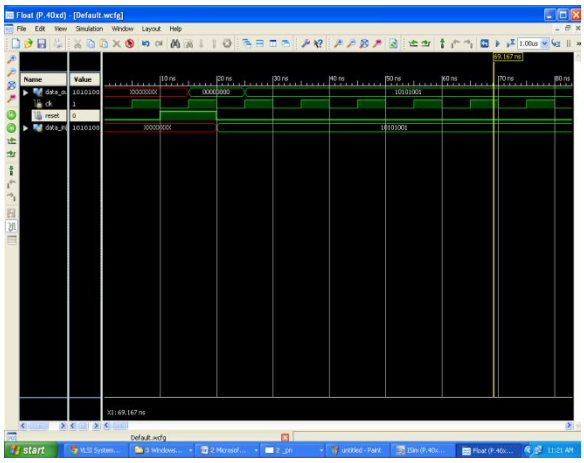


Fig 7: Output waveform

V.CONCLUSION

In this brief, the new proposed method overcomes the problems encountered in the recently presented methods for LDPC codes. The proposed method has the advantages over Prior LDPC schemes in respect of find out the error and it also decreasing the decoding time,requires less area and increasing the memory access time. The potential benefits of considering codes from combinatorial designs are guaranteed code properties.The simulation results show the correction of errors in the block of processing data of the new method soft-bit-flipping (SBF) algorithm.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.
[2] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[3] D. J. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–432, Mar. 1999.
[4] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.
[5] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981. [6] F. R. Kschischang, B. J. Frey, and H.-A.Loeliger, "Factor graphs and the sum–product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.
[7] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.
[8] J. Cho and W. Sung, "High-performance and low-complexity decoding of high-weight LDPC codes," (in Korean) *J. Korea Inf. Commun. Soc.*, vol. 34, no. 5, pp. 498–504, May 2009.
[9] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.
[10] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y.Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.
[11] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," *IEEE Commun.Lett.*, vol. 8, no. 3, pp. 165–167, Mar. 2004.
[12] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," *IEEE Commun.Lett.*, vol. 9, no. 9, pp. 814–816, Sep. 2005.