A Simplified Addition Operation Log-SPA LDPC Decoder

Po-Hui Yang, *Member, IEEE,* Jung-Chieh Chen, *Member, IEEE,* Ya-Ting Chan and Ming-Yu Lin

***Abstract*—A low hardware cost Low-Density Parity-Check (LDPC) decoder is presented in this paper. Using logical OR operation in the check nodes for the log sum-product algorithm (Log-SPA), we propose a new architecture for updating the check nodes messages. Synthesized and numerical results show that the proposed architecture achieves up to 21% total hardware reduction with fair BER performance when compared with the traditional Log-SPA decoder. Moreover, the proposed decoder also outperforms the simplest known sign-min architecture in terms of hardware complexity and BER performance.**

***Index Terms*—LDPC decoding, Sum-product algorithm.**

I. Introduction

Modern VLSI technology allows the Low-Density Parity-Check (LDPC) code [1] hardware to be realized. Remarkable error correction performance of the code introduces extensive discussions and studies [2][3]. The original LDPC decoder employs sum-product algorithm (SPA) for updating soft-information between check nodes and variable nodes, but requiring a huge hardware for performing the SPA during message update. Several methods [4]-[8] were therefore proposed to reduce hardware complexity. Logarithm field transfer, called Log-SPA [9], is the known method to replace the multipliers by adders. Under logarithm domain operation, the original multiplication operation is transferred into an addition operation. It turns out that a large number of addition devices become a hardware burden as well. Numerous hardware simplifying methodologies have been proposed in dealing with mass addition operations. For example, the parallel architecture [6] and the reorganized adder tree with the re-maps skill [8] have been proposed to improve the complexity of the hardware effectively. However, the main drawback in [6] and [8] is the large number of adders and huge look-up tables (LUT) required. On the other hand, instead of employing adders for check node operation, the sign-min algorithm employs a comparator to update message. This sign-min algorithm saves a lot of hardware, but the bit error rate (BER) performance of the sign-min algorithm is sacrificed due to the oversimplified original SPA. In this paper we try to employ some simple logic gates to replace the binary adders to perform the SPA in order to reduce hardware complexity and maintain acceptable BER performance.

This work was supported by the National Science Council of Taiwan under Grant NSC 96-2221-224-79.

Po-Hui Yang, Ya-Ting Chan and Ming-Yu Lin are with the Department of Electronic Engineering, National Yunlin University of Science & Technology, Yunlin 64002, Taiwan (phyang@yuntech.edu.tw)

Jung-Chieh Chen is with the Department of Optoelectronics and Communication Engineering, National Kaohsiung Normal University, Kaohsiung, Taiwan (corresponding author to provide phone: 886-7-7172930 ext 7718 ; fax: 07-6051146; e-mail: jcchen@nknucc.nknu.edu.tw).

II. LDPC Decoder Algorithm

Consider an LDPC code visualized by a Tanner graph consisting of M check nodes and N variable nodes as shown in Fig. 1. Let us define the set of the check node connected to the variable node, and this is denoted as $M(n)$, and the set of the variable node connected to the check node is denoted as $N(m)$. $M(n)\backslash m$ represents the set $M(n)$ with the m^{th} check node excluded, and $N(m)\backslash n$ represents the set $N(m)$ with n^{th} variable node excluded.

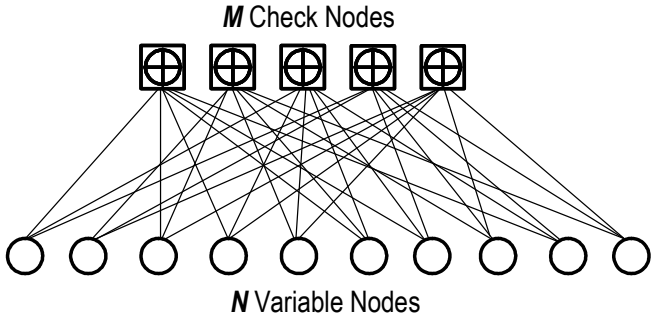

Fig. 1 Bitpartite graph of a LDPC code.

During decoding processes of the LDPC code, SPA updates the soft information iteratively between the check node and variable node. For notation simplification, let us define $\lambda_{n\to m}(u_n)$ as the soft information sent from the variable node n to check node m as

$$\lambda_{n\to m}(u_n) = \log\left\{\frac{q_{n\to m}(0)}{q_{n\to m}(1)}\right\}, \tag{1}$$

where the quantities $q_{n\to m}(x)$ is the probability information sent from the variable node n to check node m along a connecting edge of a Tanner graph, indicating $P(x_n = x)$. Also, let us define $\Lambda_{m\to n}(u_n)$ as the soft information sent from the variable node n to check node m as

$$\Lambda_{m\to n}(u_n) = \log\left\{\frac{r_{m\to n}(0)}{r_{m\to n}(1)}\right\}, \tag{2}$$

where the quantities $r_{m\to n}(x)$ is the probability information sent from the check node *m* to variable node *n* along the connecting edge of a Tanner graph, indicating $P(x_n = x)$. Next, the iteration procedures of the SPA, used in LDPC codes, can be summarized as follows:

A. Initialization

For the domain of log-likelihood, we can obtain the log-likelihood ratio (LLR) for the case of the transmitted bit u_n = 0 and u_n = 1 given the observed received bit $y_n \in \{0,1\}$ which may be corrupted by the channel noise, as follows:

$$L(u_n) = \log\frac{P(u_n = 0 \mid y_n)}{P(u_n = 1 \mid y_n)}. \tag{3}$$

The valves of $\lambda_{n\to m}(u_n)$ and $\Lambda_{m\to n}(u_n)$ are initialized to the values

$$\lambda_{n\to m}(u_n) = L(u_n) \tag{4}$$

and

$$\Lambda_{m\to n}(u_n) = 0, \tag{5}$$

respectively.

B. Check node to variable node (update check nodes):

According to the standard SPA, a check node *m* gathers all the incoming LLR messages, and evaluates the LLR message sent to the variable node *n*, which can be expressed as

$$\Lambda_{m\to n}(u_n) = \text{sign}\prod_{n'\in N(m)\setminus n}\lambda_{n'\to m}\left(\frac{u_{n'}}{2}\right) \cdot \phi^{-1}\left(\sum_{n'\in N(m)\setminus n}\phi\left[\lambda_{n'\to m}\left(\frac{u_{n'}}{2}\right)\right]\right), \tag{6}$$

where

$$\phi(x) = -\log[\tanh(x/2)] = \log\left(\frac{e^x+1}{e^x-1}\right) \tag{7}$$

C. Variable node to check node (update variable node):

According to the standard SPA, a variable node *n* passes the LLR message to all the check nodes connected to it, which can be expressed as

$$\lambda_{n\to m}(u_n) = L(u_n) + \sum_{m'\in M(n)\setminus m}\Lambda_{m'\to n}(u_n) \tag{8}$$

D. Check stop criterion

The overall LLR message of a variable node n indicating the probability to decode variable node *n* to 1 or 0 can be obtained by adding up all the incoming LLR messages to the variable node n as

$$\lambda_n(u_n) = L(u_n) + \sum_{m\in M(n)}\Lambda_{m\to n}(u_n). \tag{9}$$

After each iteration, a hard decision on variable node *n* is made, that is, the variable *n* is decoded as "one" when $\lambda_n(u_n) \ge 0$, and decoded as "zero" otherwise. If all the decision bits constitute a valid codeword, the algorithm stops and outputs the decoding result. Otherwise, the algorithm repeats Steps *B-D*.

III. LOG-SPA LDPC DECODER HARDWARE

Hardware structures of the traditional Log-SPA for the variable node and the check node are shown in Fig. 2 and Fig. 3, respectively. Since the hardware complexity of the check node is more complex, we will focus on the hardware architecture of the check node. Check node can be further implemented by two independent operational parts, the sign and addition part [6].

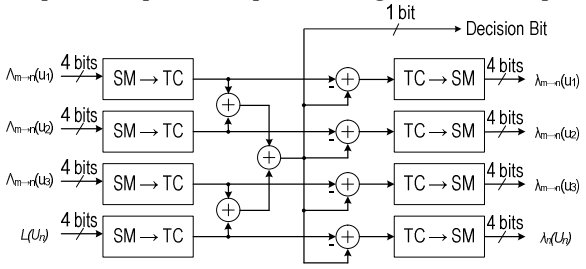

Fig. 2 Data path of traditional variable node (sign-magnitude, SM, two's complement, TC).

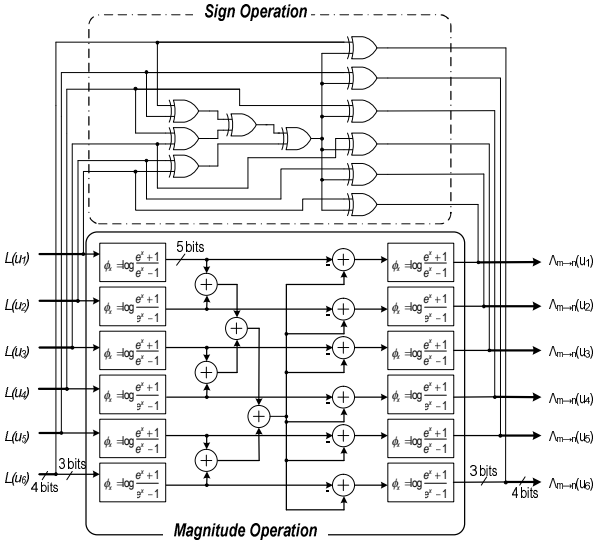

Fig. 3 The traditional Log-SPA check node architecture.

Before feeding the LLR values for performing the SPA, we have to confine the input range followed by a quantizer rounding off the input values to the nearest value in a set of *q* quantum levels. Table I shows the 4-bit sign binary numbers quantization table. In contrast, to execute the message update in the check node in (6), we have to map input values through hyperbolic tangent (*tanh*) function for further calculation. Next, the calculated value via (8) for the variable node message update has to be converted. Hyperbolic tangent mapping function can be implemented by a look-up table. We can construct the input and output mapping table as Table II and Table III, respectively, based on the characteristic curve of (7) shown in Fig. 4. It is worth pointing out that the bit-length of the quantized LLR value is a trade-off between the hardware complexity and the BER performance.

TABLE I
QUANTIZATION RANGE

LLR	Q-LLR
$<000.101_2$ ($<0.625_{10}$)	000_2 (0_{10})
$000.101_2 \sim 001.101_2$ ($0.625_{10} \sim 1.625_{10}$)	001_2 (1_{10})
$001.101_2 \sim 010.101_2$ ($1.625_{10} \sim 2.625_{10}$)	010_2 (2_{10})
$010.101_2 \sim 011.101_2$ ($2.625_{10} \sim 3.625_{10}$)	011_2 (3_{10})
$>=011.101_2$ ($>=3.625_{10}$)	100_2 (4_{10})

TABLE II
3-BIT TO 5-BIT LOOK-UP TABLE

Q-LLR	ϕ(X)
000_2 (0_{10})	1.1111_2 (1.9375_{10})
001_2 (1_{10})	0.1101_2 (0.8125_{10})
010_2 (2_{10})	0.0100_2 (0.2500_{10})
011_2 (3_{10})	0.0001_2 (0.0625_{10})
100_2 (4_{10})	0.0000_2 (0.0000_{10})

TABLE III.
5-BIT TO 3-BIT LOOK-UP TABLE

ϕ (X)	LLR
$1.1111_2 \sim 1.1000_2$ ($1.9375_{10} \sim 1.5000_{10}$)	000_2 (0_{10})
$1.1000_2 \sim 0.1000_2$ ($1.5000_{10} \sim 0.5000_{10}$)	001_2 (1_{10})
$0.1000_2 \sim 0.0011_2$ ($0.5000 \sim 0.1875$)	010_2 (2_{10})
$0.0011_2 \sim 0.0001_2$ ($0.1875_{10} \sim 0.0625_{10}$)	011_2 (3_{10})
$0.0001_2 \sim 0.0000_2$ ($0.0625_{10} \sim 0.0000_{10}$)	100_2 (4_{10})

SPA should be performed in the logarithm domain to avoid intensive multiplicative operation while performing it. Multiplicative operation can be replaced by an additive operation. From the hardware point of view, speed and area bottlenecks of the LDPC decoder are also transferred from multipliers to adders. To cope with the addition bottleneck, a quasi binary-weighting mapping table shown in Table II is proposed. We then use a simple logic gate circuit to replace the traditional adders to update the message in the check node. In the next section, we will explain the proposed circuit in detail.

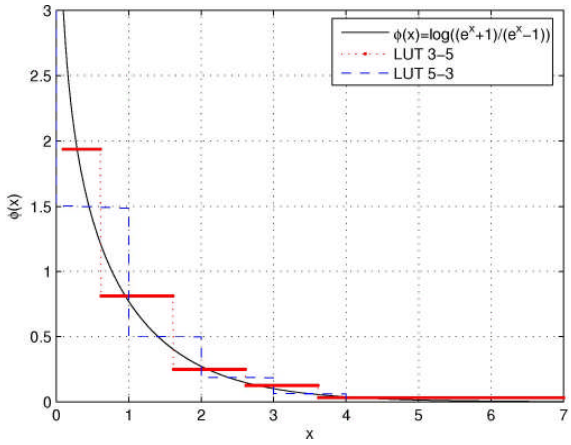

Fig. 4 Quantization curve for lookup-tables.

IV. ADDER FREE CHECK NODE ARCHITECTURE

As shown in Fig. 3, we have to map the input value through Table II to obtain the desired output value before we feed the input LLR values into the check node for addition operation. Observing Table II, the mapping relationship can be approximated as a "quasi binary-weighting relationship." Inspired by this quasi binary-weighting relationship in Table II, we propose using OR operations to replace the original adder in the check node. The proposed OR operation circuit architecture is shown in Fig. 5.

Performance of the proposed OR operation to carry out the SPA has been evaluated by numerical experiment. The code used for simulation is (1008, 3, 6) regular LDPC code [10]. We use two algorithms, the original SPA, LUT 3-5 [6], and the sign min algorithm [4] to demonstrate the performance improvement of the proposed scheme for comparison. The simulation results are shown in Fig. 6, where the BER performance versus signal-to-noise ratio (SNR) is compared among SPA, LUT 3-5 [6], the sign min algorithm [4], and the proposed OR operation. The maximum iteration number we set in the simulation is 80. Figure 6 shows that the BER performance of the proposed OR operation outperform the sign-min algorithm.

Next, let us compare the hardware complexity issue. The hardware comparison is shown in Fig. 7 based on the 0.18 μm standard cell library. From Fig. 7, we find that the OR operation architecture has 21% hardware reduction comparing with the traditional LUT 3-5 architecture. This is also since the proposed architecture uses simple gates to perform the SPA. Our proposed OR operation architecture can therefore work up to a 100 MHz decoding speed. Table IV shows the speed comparisons.

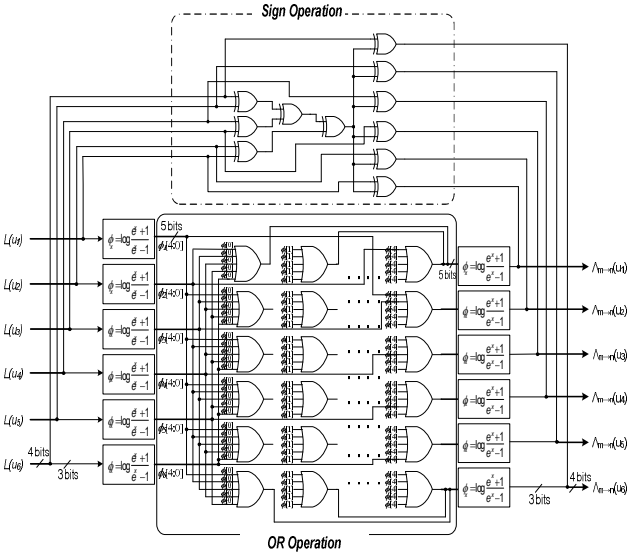

Fig. 5 Proposed OR operation check-node architecture.

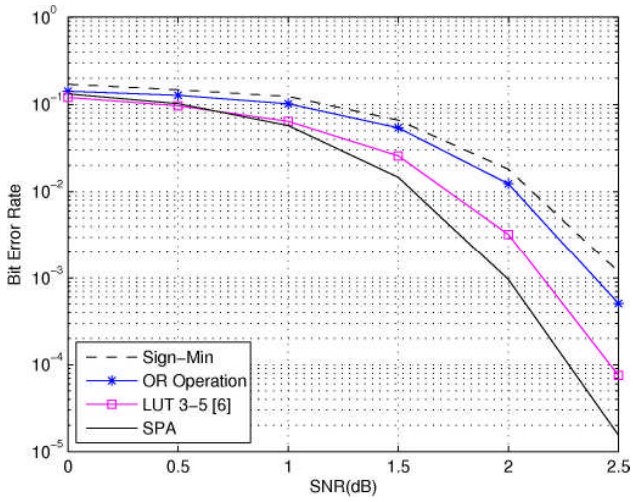

Fig. 6 BER performances for (1008, 504) LDPC code with 80 decoding iterations.

TABLE IV
SPEED COMPARISONS

	Block Length	Technology	Speed
Log-SPA [6]	1024	0.16μm	64 MHz
Sign-Min [7]	1008	0.18μm	90 MHz
This Work	1008	0.18μm	100 MHz

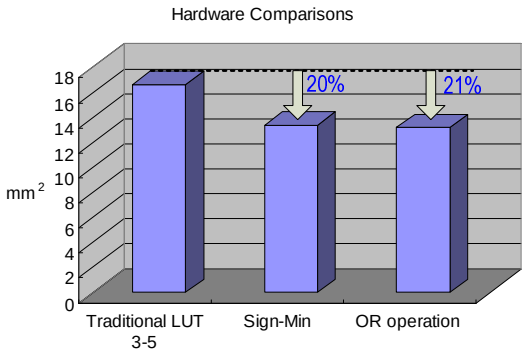

Fig. 7 Hardware comparisons using 0.18μm standard cell library.

V. CONCLUSION

We have proposed the OR operation to replace the addition operation in the check nodes for the Log-SPA. Comparing with the traditional Log-SPA architecture, the new circuit structure has already made a hardware reduction up to 21% effectively. Furthermore, when comparing with the simplest sign-min architecture [7], our new architecture has lower hardware cost, higher decoding speed, and better BER performance.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check code", *IRE Trans. Inform.Theory*, vol.IT-8, pp. 21-28, Jan. 1962.

[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices", *IEEE Trans. Inform. Theory*, vol. 45, pp. 399-43 1, Mar. 1999.

[3] S. -Y. Chung, G. D. Forney, Jr., T, J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit", *IEEE Commun. Lett*, vol. 5, pp. 58-60, Feb. 2001.

[4] X. -Y Hu, E. Eleftheriou, D. -M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes", *Proc. IEEE Globecom*, pp. 1036-1036E, Nov. 2001.

[5] S. Papaharalabos, P. Sweeney, B. G. Evans, G. Albertazzi, A. Vanelli-Coralli and G. E. Corazza, "Performance evaluation of a modified sum-product decoding algorithm for LDPC codes", *IEEE International Symposium on Wireless Communication Systems*, Page(s):800 – 804, Sept. 2005.

[6] A. J. Blanksby and C. J. Howland, "A 690-mW 1 Gb/s 1024-b rate-1/2 low-density parity-check code decoder", *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[7] C. -C Lin, K. –L Lin, H. –C Chang and C. –Y Lee, "A 3.33Gb/s (1200,720) low-density parity check code decoder", *Proc.of the 31st European IEEE Solid-State Circuits Conference*, Page(s): 211 – 214, Sept. 2005.

[8] Y. Zhang, Z. Wang and K. K. Parhi, "Efficient high-speed quasi-cyclic LDPC decoder architecture," *IEEE ACSSC*, Vol.1, pp. 540-544, Nov. 2004.

[9] W. E. Ryan, *An introduction to LDPC codes*, in CRC Handbook for coding and signal processing for recording systems (B. Vasic ed.), CRC Press, 2004.

[10] D. J. C. Mackay, *Online database of low-density parity-check codes*, available at http://www.inference.phy.cam.ac.uk/mackay/CodesFiles-.html.

[11] S. Tong, P. Wang, D. Wang and X. Wang, " Box-minus operation and application in sum-product algorithm," *IEE Electronics Letters*, vol. 41, pp. 197-198, Feb. 2005.