Design and Implementation of High Code-Rate LDPC Decoder based on CP-PEG Code Construction

Kao-Shou Lin, Chih-Lung Chen, Hsie-Chia Chang, Chen-Yi Lee

***Abstract*—This brief present a area-efficient LDPC decoder architecture which is suitable for high code-rate LDPC code. A high code-rate $15/16$ (2048, 1920) irregular LDPC code is constructed by CP-PEG algorithm as a design example. The resulting large check node degrees become the implementation bottleneck. To design such high code-rate LDPC decoder, several design techniques are employed, including variable-node-centric sequential scheduling to reduce iteration number, single piplelined decoder architecture to lessen the message storage memory size, as well as optimized check node unit to further compress the register number. Overall 73% message storage memory is saved as compared with traditional architecture. After fabrication in 90nm 1P9M CMOS technology, the test decoder chip could achieve maximum 11.5 Gbps throughput under 1.4V supply voltage with core area of 3.78 mm^2. The energy efficiency is only 0.033 nJ/bit with 5.77 Gbps at 0.8V to meet throughput requirements of IEEE 802.15.3c .**

***Index Terms*—LDPC decoder, high code-rate, sequential scheduling.**

I. Introduction

Low-density parity-check (LDPC) code is a famous error control code with near Shannon limit performance [2] and can be described by its parity-check matrix **H**. The rows and columns of **H** are mapped to check nodes and variable nodes of a bipartite graph, on which the belief-propagation (BP) algorithm exchange messages between nodes iteratively to decode LDPC codes [3]. The message exchanging order between nodes is called scheduling, which will influence the convergence speed of the decoding algorithm. In standard BP algorithm, simultaneous update of all check node messages or variable node messages is named as *flooding scheduling*. Alternatively, the layered BP algorithm [4] [5] preforming message update along different check node groups is a method of *check-node-centric sequential scheduling* (CSS). Researches have revealed that CSS could reduce maximum iteration to approximate half of the standard BP with similar performance.

Recently, LDPC codes adopted in high-throughput systems have high code-rate property to increase channel efficiency. However, the introduced large check node degree dc will cause implementation difficulties. For example, the check node degree of (2048, 1723) LDPC code adopted in IEEE 802.3an [6] equals 32, leading to routing difficulty and low chip density. Even though the CSS could reduce the iteration number, the throughput is still degraded due to long critical path of check node unit (CNU). The situation will become worse for the (1440, 1344) LDPC code of IEEE 802.15.3c [7] with $dc = 45$. Simply adding pipelined staged in CNU or decreasing parallelism may not be a suitable solution to solve the bottleneck. Based on the *variable-node-centric sequential scheduling* (VSS) [8], using a (2048,1920) code-rate $R = 0.9375$ LDPC code with $dc = 46$ constructed by circulant permutation progressive edge-growth (CP-PEG) algorithm [9] as a design example, the proposed decoder could provide a high-throughput and area-efficient solution to the high code-rate LDPC code.

The remainder of this brief is organized as follows. Section II introduces the issues in permutation and partition parity check matrix. Section III presents detail of optimized check node units. Implementation result are shown in section IV. The conclusion is given in section V.

II. Issue in single pipelined architecture

A. Variable-node-centric Sequential Scheduling

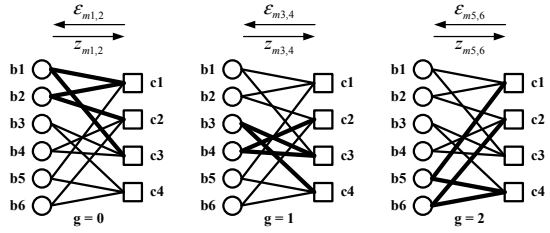

Fig. 1. Illustration of VSS with $N = 6$, $G = 3$ and $N_G = 2$.

To simply explain the VSS approach, a bipartite graph in Fig. 1 is used for an example. Let $\varepsilon_{m,n}$ denote the messages sent from check node m to variable node n. Let $z_{m,n}$ denote the messages sent in opposite direction. In standard BP, all of $\varepsilon_{m,n}$ are updated in parallel, then all of $z_{m,n}$ are updated in parallel in one iteration. In VSS approach, the N bits of a codeword are partitioned into G groups, so each group contains $N/G = N_G$ bits. $\varepsilon_{m,n'}$ and $z_{m,n'}$, where n' are located in g-th group, are updated sequentially. In addition, it count one iteration when all group are updated.

B. Parity Check Matrix Partition and Permutation

In VSS approach, the essential design parameter in single pipelined architecture is the number of group G which the par-

This paper was presented in part at the IEEE European Solid-State Circuits Conference (ESSCIRC), Athens, Greece, September 2009 [1].

The authors are with the Department of Electronics Engineering and Institute of Electrics, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: eeeericlin.ee96g@g2.nctu.edu.tw)

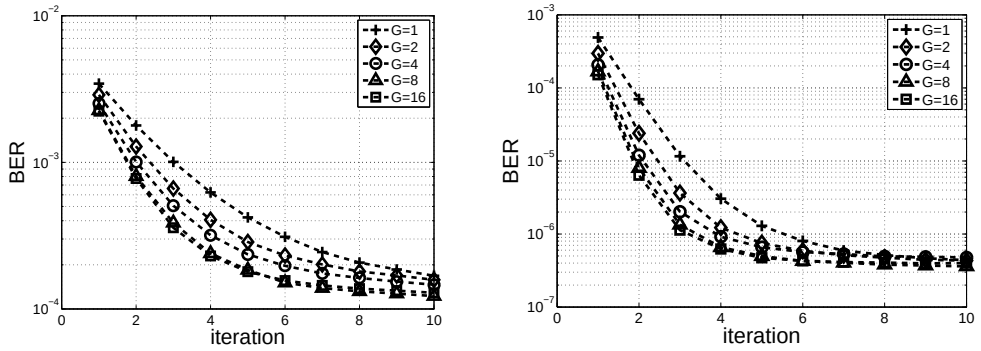

Fig. 2. convergence speed of (2048,1920) LDPC code by using VSS decoding algorithm in additive white gaussian noise (AWGN) channel with binary phase shift keying (BPSK) modulation with G set to $1, 2, 4, 8$ and 16 respectively. (a) SNR=5.5. (b) SNR=6.

TABLE I
SINGLE PIPELINED ARCHITECTURE DESIGN PARAMETER

CNU Number	VNU Number	Routing Network
M	N/G	$2 \times N/G$ G-input Mux

ity check matrix should be partitioned into. Both of throughput and hardware complexity are affected by G. The throughput of single pipelined architecture could be calculated in (1).

$$throughput = \frac{f_{clk} \times N \times R}{(I+1) \times G} \quad (1)$$

f_{clk} is the clock frequency, and I is the iteration number. From (1), the throughput will be degraded by a factor G, and the throughput is affected by decoder's convergence speed $\propto 1/I$. Fig. 2 shows the bit error rate (BER) versus iteration at $SNR = 5$ and 6. The convergence speed of decoders with G set to 8 and 16 are almost the same, leading less than two times speed-up. On the other hand, hardware complexity will be significantly reduced when G increase, because only N/G VNUs are required in single pipelined architectures as shown in table I.

However, our target throughput is 10Gbps in early design stage. By evaluating the performance curve and (1), $G = 4$ and $I = 4$ is used in our design.

As shown in Fig. 3, the parity-check matrix **H** is divided and permuted into 4 submatrices (**H** $=$ [**H1 H2 H3 H4**]). Because the check node to variable node message is on-the-fly calculated, $16 \times p$ $(64 \times p \div 4)$ VNU's should be used in single pipelined architecture. To avoid redundant VNUs appearing when updating different groups, each submatrix is permuted to contain same variable node degree distribution. Therefore, total $16 \times p$ VNUs in the decoder are composed of $4 \times p$ VNU4s, $6 \times p$ VNU3s and $6 \times p$ VNU2s. In addition, because the VNUs are reused for updating different group of messages, the 4 inputs MUX (MUX4) are needed for different group of messages selection. By arranging the CP matrix (shaded) with same shift amount in the same position of each submatrix, the connections between correspond CNUs and VNUs will be the same when updating different group. Therefore, the MUX could be reduced, and the routing and control could be further simplified. With this technique, about 15% MUX4 and 25% MUX2 could be reduced. Based on the same concept, the MUX could be further reduced by permuting inner CP matrix to make same shift amount. The method could be done by adding offset to original shift amount and reordering correspond input channel value. As a result, 25% mux4 could be further reduced from 15%.

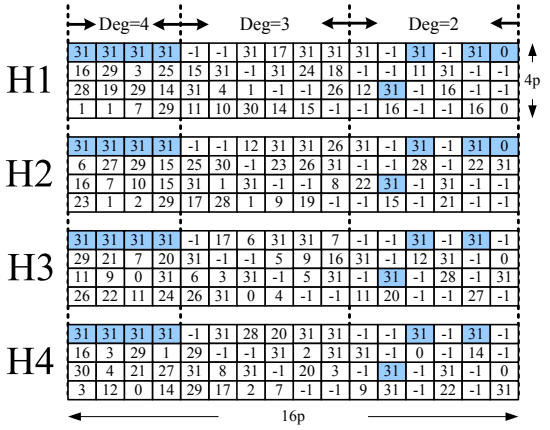

Fig. 3. Permuted parity-check matrix of (2048, 1920) LDPC code.

III. OPTIMIZED CHECK NODE UNIT

In this section, we detail the CNU design which support single pipelined architecture in VSS.

A. Accumulative Sorter

In standard BP, the computation of CNU can be divided into magnitude part and sign part. Thus, the messages sent from VNU are converted from two's complement format to sign-magnitude format for efficient computation of CNU. If min-sum decoding is used, the magnitude part of check node to variable node messages will be either minimum or second minimum of messages sent from neighboring VNUs. Therefore, dc inputs sorter is commonly used to implement magnitude part of CNU.

From the characteristics of VSS, the check node to variable node messages in VSS could be computed in accumulative way. Similar to CNU in standard BP, the CNU in VSS is composed of accumulative sorter and accumulative sign computation unit. Let $\alpha_{m,g}^{(i)}$, $\beta_{m,g}^{(i)}$ denote the local minimum and local second minimum of messages sent from variable

nodes in the g-th group to one specific check node m at i-th iteration, which are:

$$\alpha_{m,g}^{(i)} = \min_{\substack{n' \in N(m) \\ g \cdot N_G \le n' \le (g+1) \cdot N_G - 1}} \left\{ \left| z_{mn'}^{(i)} \right| \right\} \quad (2)$$

$$\beta_{m,g}^{(i)} = \min_{\substack{n' \in N(m) \backslash argmin(\alpha_{m,g}^{(i)}) \\ g \cdot N_G \le n' \le (g+1) \cdot N_G - 1}} \left\{ \left| z_{mn'}^{(i)} \right| \right\} \quad (3)$$

Let $\varepsilon_{m,g}^{(i)}$, $\delta_{m,g}^{(i)}$ denote the global minimum and global second minimum of sorted messages sent from variable nodes to one specific check node m at i-th iteration, which are:

$$\varepsilon_{m,g}^{(i)} = \min \left\{ \left\{ \alpha_{m,j}^{(i)} \right\}_{0 \le j \le g}, \left\{ \alpha_{m,k}^{(i-1)} \right\}_{G > k > g} \right\} \quad (4)$$

$$\gamma_{m,g}^{(i)} = \min_{j,k \ne argmin(\varepsilon_{m,g}^{(i)})} \left\{ \left\{ \alpha_{m,j}^{(i)} \right\}_{0 \le j \le g}, \left\{ \alpha_{m,k}^{(i-1)} \right\}_{G > k > g} \right\} \quad (5)$$

$$\delta_{m,g}^{(i)} = \min \left\{ \gamma_{m,g}^{(i)}, \left\{ \beta_{m,j}^{(i)} \right\}_{0 \le j \le g}, \left\{ \beta_{m,k}^{(i-1)} \right\}_{G > k > g} \right\} \quad (6)$$

Fig. 4 illustrates the magnitude part of CNU's computation in VSS, which is an accumulative sorter composed of a local sorter and a global sorter. For m-th check node , $\alpha_{m,g}$ and $\beta_{m,g}$ in (2) and (3) are computed by local sorter. Notice that $\alpha_{m,g}$ and $\beta_{m,g}$ in each group should be stored for computing $\varepsilon_{m,g}$ and $\delta_{m,g}$ in (4) and (6) by global sorter. Then the $\varepsilon_{m,g}$ and $\delta_{m,g}$ could be used to update variable nodes in g-th group.

For our proposed CP-PEG LDPC codes with $dc = 46$, the VSS approach with $G = 4$ could divide 46 inputs of the CNU into four part. Because 46 could not be divisible by 4, the number of messages need to be computed in different groups will be 11 or 12. This could be handled by little extra circuit which determines the number of messages which should be computed by CNU when updating different groups.

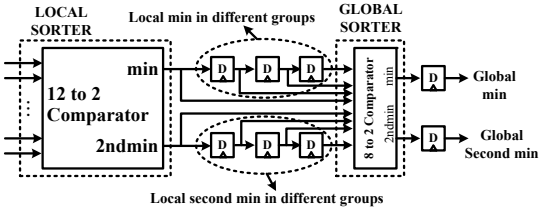

Fig. 4. Accumulative sorter

B. Accumulative Sign Computation Unit

Fig. 5 shows the sign part of CNU in VSS. Similarly, the sign part of CNU in VSS can be computed in an accumulative way like the accumulative sorter. Here, XORN represent N inputs exclusive-or gate. XOR12 is used to compute the local sign of $\lceil d_c/4 \rceil$ input messages's signs which are queued in each cycle, and XOR4 is used to compute the global sign of 4 local signs. When the global sign is available, the $\lceil d_c/4 \rceil$ messages's signs are computed with global sign by XOR and sent to VNUs in g-th group .

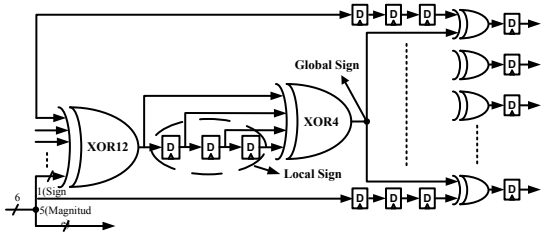

Fig. 5. accumulative sign computation unit

C. Reduced Memory Accumulative Sorter 1

To further reduce the storage overhead of each group, we proposed a Reduced Memory Accumulative Sorter 1 (RMAS1) as shown in Fig. 6. The basic idea is to only store one group of local minimum and local second minimum instead of $G-1$ group. Some extra control circuits are needed to open or close the feedback loop in Fig. 6. This sorter architecture is beneficial since the complexity reduction of storage registers and global sorters is higher than the overhead of control circuits. The detail algorithm of RMAS1 is listed below. The open loop operation can be implemented by adding extra control circuit which make the value of feedback look invalid in comparison.

Algorithm: Reduced Memory Accumulative Sorter 1

1: **Initialization**
2: inner feed back loop closed
3: outer feed back loop open
4: find global min and global second min
5: **Decoding**
6: inner feed back loop open
7: **if**(global min in current Group **or** current Group $== 4$)
8: global min outer feed back loop open
9: **else**
10: global min outer feed back loop closed
11: **if** (global second min in current Group **or** current Group$== 4$)
12: global second min outer feed back loop open
13: **else**
14: global second min outer feed back loop closed

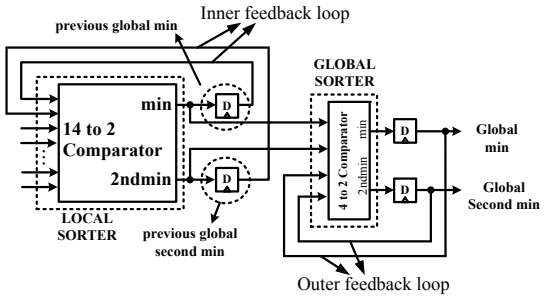

Fig. 6. Reduced storeage accumulative sorter

D. Reduced Memory Accumulative Sorter 2

$\alpha_{m,g}$ and $\beta_{m,g}$ of 4 group are required to be computed and stored for the guarantee of $\varepsilon_{m,g}$ and $\delta_{m,g}$ correctly computed by global sorter. For the same purpose to reduce storage overhead, we proposed a Reduced Memory Accumulative Sorter 2 (RMAS2) as shown in Fig. 8. The basic idea is to only store $\alpha_{m,g}$ in each group, and use 4-to-2 sorter to compute

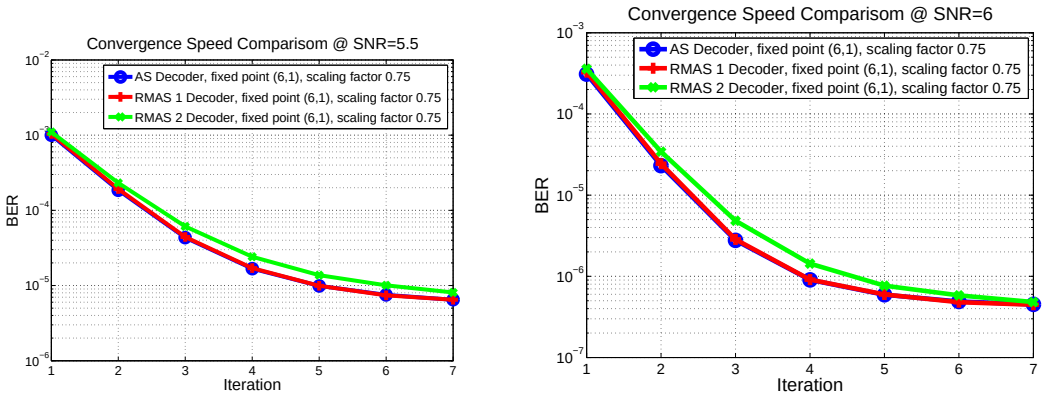

Fig. 7. Convergence speed of three type of decoder with different CNU architectures in the additive white Gaussian noise (AWGN) channel and BPSK modulation. (a) SNR=5. (b) SNR=6.

global minimum and global second minimum. In fact, the global minimum equals $\varepsilon_{m,g}$, but the global second minimum equals $\gamma_{m,g}$ in (5) instead of $\delta_{m,g}$ in (6).

Clearly, global minimum will be correctly sorted in comparison with original accumulative sorter; nevertheless, "true" global second minimum could be sorted incorrectly when the global minimum and global second minimum are located in the same group of variable nodes, because the "true" global second minimum will be dropped after passing through local sorter.

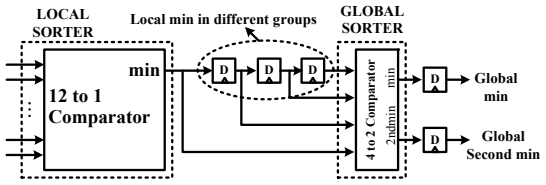

Fig. 8. Reduced storeage accumulative sorter 2

E. Trade-off between different Sorter Architectures

Three type of decoder are implemented in same single pipelined architecture with different type of CNU architecture. AS decoder, RMAS1 decoder and RMAS2 decoder denote the decoder which's CNUs are implemented by accumulative sorter, Reduced Memory Accumulative Sorter 1 and Reduced Memory Accumulative Sorter 2 respectively. The sign part of CNU in three decoders are implemented in the same architecture (i.e., accumulative sign computation unit).

To achieve performance of floating point simulation, the input LLRS are quantized to 6-bit (5-bit integer and 1-bit decimal fraction) , and hardware-friendly scaling factor is set to 0.75. Fig. 7 shows the convergence speed of three kind of decoder. One can observe that the convergence speed of RMAS1 decoder is almost equivalent to original AS decoder, but convergence speed of RMAS2 decoder is slightly lower than AS decoder.

To investigate the hardware complexity, each decoder is synthesized by Synopsys design compiler with same tickle file. The synthesis result (whole decoder) is shown in table II; Compared to original AS decoder, RMAS1 decoder provide less than 10% gate count reduction, and RMAS2 decoder provide about 20% gate count reduction.

TABLE II
COMPARISON BETWEEN DIFFERENT SORTER ARCHITECTURES

Architecture[1]	AS Decoder	RMAS1 Decoder	RMAS2 Decoder
Gate Count[2]	727k	661k	570k

[1] Synthesis result by utilizing UMC 90-nm 1P9M CMOS technology
[2] The clock period is set to 9ns during synthesis

IV. Implementation Result and Comparison

After evaluating the trade-off between different sorter architecture, RMAS1 decoder is chosen to be implemented.

A. Comparison with Conventional Architecture

In conventional two-stage pipelined architecture, both $z_{mn}^{(i)}$ and $\varepsilon_{mn}^{(i)}$ messages are stored in registers or memory. As reported in [1], overall register reduction of proposed architecture is 73%, leading to the following advantages: fewer registers, higher utilization of functional units, and reduced complexity. In addition, a conventional compare-select-tree based 46 inputs sorter is implemented to compare the hardware complexity of accumulative sorter. Two sorter architecture are synthesized at different delay by utilizing UMC 90-nm 1P9M CMOS technology. As shown in Fig. 9, accumulative sorter occupied less than 50% gate count compared to 46 inputs sorter at diferent delay.

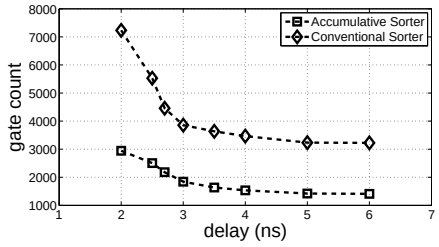

Fig. 9. gate count of different sorter architecture.

TABLE III
COMPARISON WITH STATE-OF-THE-ART

	RMAS1 decoder*		Tinoosh**[10]		Zhang*[11]		Yamagishi***[12]
CMOS technology [nm]	90		65		65		90
Code Spec	(2048, 1920)		(2048,1723)		(2048,1723)		(1400,1344)
Code Rate	0.9375		0.84		0.84		0.9333
Check Node Degree	46		32		32		45
Core Area [mm^2]	3.84		4.84 (9.68[a])		5.35 (10.7[a])		0.5
Gate Count	708k		690k		N.A		409k
Iteration	4		15		8		8
Input Quantization	6 bits		5 bits		4 bits		4 bits
Supply Voltage(V)	1.4	0.8	1.3	0.7	1.2	0.7	1.2
Clock Frequency [MHz]	120	60	195	35	700	100	216
Throughput [Gbp/s][ab]	11.5	5.77	36.3 (25.7[a])	6.5 (4.6[a])	14.9 (10.53[a])	2.1 (1.48[a])	6.45
Power [mW]	1037	191.2	1359 (3844[a])	62 (175[a])	2800 (7919[a])	144 (407[a])	86 (307[b])
Energy Efficiency[nJ/bit][a]	0.09	0.033	0.15	0.038	0.752	0.275	0.047
Hardware Efficiency[Gb/s/mm^2] [ab]	2.99	1.5	2.65	0.48	0.98	0.13	12.9

[a] Technology scaling to 90 nm CMOS assuming: $A \propto 1/s^2$, $t_{pd} \propto 1/s$, and $P_{dyn} \propto 1/s^3$.
[b] Without early termination
* Measurement Result
** Post Layout Result
*** Synthesis Result

B. Implementation Result and Discussion

Table III summarizes the key characteristics of the implemented RMAS1 decoder in 90-nm 1P9M CMOS technology. The core occupied 3.84 mm^2 area with 68% utilization. The die photo is shown in Fig. 10, and the distribution of CNUs and VNUs is auto-determined by APR tool.

The throughput is calculated without early termination for comparison under same condition. The throughput is highest in [10] because of the fully parallel architecture. However, hardware efficiency of our decoder is higher than [10] and [11] even though our quantization bit width is higher. The decoders in [10] and [11] occupy large area because fully-parallel VNUs architecture, which dominate the decoder's complexity for higher code-rate LDPC code, are adopted in both design.

In fact, the architecture in [12] is very similar to our design: single pipeline stage of the decoder, partial parallel VNUs, and reduced complexity check node unit. The difference lies on two aspects:(a) VSS is adopted in our design which could result in higher convergence speed, and (b) routing network in [12] is static connection between CNUs and VNUs by using the characteristic of the code structure of 802.15.3c LDPC code. However, if 802.15.3c LDPC code are mapped to our proposed architecture, the routing network could be static between CNU and VNU by shift the values circularly between CNUs based on the similar technique in [12] and [13].

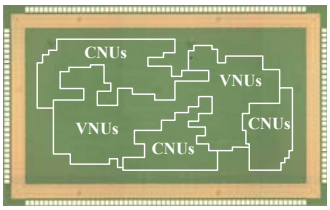

Fig. 10. Chip micrograph.

V. CONCLUSION

In this brief, we discussed the design of a high code-rate LDPC decoder based on CP-PEG code construction. Partition and permutation of the parity check matrix was introduced to eliminate the mux between CNU and VNU. Three type of low complexity CNUs in VSS, which make single pipelined architecture feasible, was also introduced. In conclusion, our proposed techniques and single pipelined architectures was suitable for high code-rate LDPC code, leading area efficient and high energy efficiency decoder.

ACKNOWLEDGMENTS

The authors would like to thank UMC, NCTU Si2 Lab, NCTU VDA Lab, NCTU-MTK Research Center, and CIC for their assistance.

REFERENCES

[1] C.-L. Chen, K.-S. Lin, H.-C. Chang, W.-C. Fang, and C.-Y. Lee, "A 11.5-gbps ldpc decoder based on cp-peg code construction," sep. 2009, pp. 412 –415.
[2] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[3] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.
[4] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. on VLSI Systems*, vol. 11, no. 6, pp. 976–996, Dec. 2003.
[5] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of ldpc codes," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS'04)*, Oct. 2004, pp. 107–112.
[6] *Part 3: carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specificaions*, IEEE Std. P802.3an-2006, Sept. 2006.
[7] *Part 15.3: wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless personal area networks (WPANs)*, IEEE Std. P802.15.3c-DF8, 2009.
[8] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, Feb. 2005.
[9] Y. K. Lin, C. L. Chen, Y. C. Liao, and H. C. Chang, "Structured LDPC codes with low error floor based on peg tanner graphs," in *IEEE Int. Sympo. Circuits and Systems (ISCAS'08)*, May 2008, pp. 1846–1849.
[10] T. Mohsenin, D. Truong, and B. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in ldpc decoders," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 57, no. 5, pp. 1048 –1061, may. 2010.
[11] Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "A 47 gb/s LDPC decoder with improved error rate performance," in *Proc. Int. Sympo. VLSI Circuits (SOVC'09)*, June 2009.
[12] H. Yamagishi and M. Noda, "High throughput hardware architecture for (1440,1344) low-density parity-check code utilizing quasi-cyclic structure," *Turbo Codes and Related Topics, 2008 5th International Symposium on*, pp. 78 –83, sep. 2008.

[13] J. Sha, Z. Wang, M. Gao, and L. Li, "Multi-gb/s ldpc code design and implementation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 2, pp. 262 –268, feb. 2009.