Multi-Gb/s LDPC Code Design and Implementation

Jin Sha, Zhongfeng Wang, *Senior Member, IEEE*, Minglun Gao, and Li Li

***Abstract*—Low-density parity-check (LDPC) code, a very promising near-optimal error correction code (ECC), is being widely considered in next generation industry standards. The VLSI implementation of high-speed LDPC decoder remains a big challenge. This paper presents the construction of a new class of implementation-oriented LDPC codes, namely *shift-LDPC codes*. With girth optimization, this kind of codes can perform as well as computer generated random codes. More importantly, the decoder can be efficiently implemented to obtain very high decoding speeds. In addition, more than 50% of message memory can be generally saved over conventional partially parallel decoder architectures. We demonstrate the benefits of the proposed techniques with an application-specific integrated circuit (ASIC) design (in 0.18-μm CMOS) for a 8192-bit regular LDPC code, which can achieve 5 Gb/s throughput at 15 iterations.**

***Index Terms*—Error correction codes (ECC), low-density parity-check (LDPC) codes, Min-Sum algorithm, parallel processing, VLSI.**

I. Introduction

ERROR correction codes (ECC) are widely applied in modern digital communication systems. Turbo codes and low-density parity-check (LDPC) codes [1] are the two most popular ECC that have near the Shannon limit performance. Since the rediscovery of LDPC codes [2], significant improvements have been experienced on the design and analysis of LDPC codes. However, the realization of a high speed LDPC decoder implementation still remains a big challenge and is a crucial issue determining how well we can exploit the unmatched merits of the LDPC codes in practical applications.

A satisfying LDPC decoder usually means: good error correction performance, low hardware complexity and high throughput. To implement the decoder directly in its inherent parallel manner may get the highest decoding throughput. But for large codeword lengths (e.g., larger than 1000 bits), to avoid routing conflict, the complex interconnection may take up more than half of the chip area [3]. Both serial and partly parallel VLSI architectures are well studied nowadays [4]–[8]. However, none of these approaches are good for very high throughput (i.e., multi-gigabits per second) applications.

Manuscript received August 13, 2007; revised November 13, 2007 and January 21, 2008. First published November 25, 2008; current version published January 14, 2009. This work was supported in part by the High-Tech Foundation of Jiangsu Province of China under Grant BG2005030 and by the National Nature Science Foundation of China under Grant 90307011.

J. Sha, M. Gao, and L. Li are with Institute of VLSI design, KLAPEM, Nanjing University, Nanjing 210093, China (e-mail: jeanshajin@gmail.com; gaominglun@nju.edu.cn; lili@nju.edu.cn).

Z. Wang was with School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA (e-mail: zwang@ece.orst.edu).

Digital Object Identifier 10.1109/TVLSI.2008.2002487

In this paper, we propose a new LDPC decoder architecture targeting for multi-gigabits per second applications. The architecture has three major merits.

1) *Memory Efficient.* By exploring the special features of the Min-Sum [9], [10] decoding algorithm, the proposed architecture can generally save over fifty percent of message memory over the conventional design for high rate codes.
2) *Highly Parallel.* The architecture can normally exploit more level of parallelism in the decoding algorithm than conventional partially parallel decoder architectures [4], [5].
3) *Low Routing Complexity.* Through introducing a special structure in the parity check matrix, the complex message passing between variable nodes and check nodes can be alleviated by the regular communication between check nodes.

The codes suited for this kind of decoder architecture are named *shift-LDPC codes*.

The remainder of this paper is organized as follows. In Section II, a brief review of LDPC codes is given. In Section III, the shift LDPC code construction is discussed. The decoder architecture is presented in Section IV. In Section V, an application-specific integrated circuit (ASIC) implementation of an 8192-bit rate-7/8 LDPC code based on the proposed architecture is presented. Section VI gives experimental results and concludes this paper.

II. Review of LDPC Codes

A. LDPC Decoding Algorithms

A $(N, K)(j, k)$ LDPC code has a Tanner graph with N variable nodes and M check nodes in which all the variable nodes have degree j and all the check nodes have degree k. The typical LDPC decoding algorithm is the Sum-Product (or belief propagation) algorithm [2]. After variable nodes are initialized with the channel information, the decoding messages are iteratively computed by all the variable nodes and check nodes and exchanged through the edges between the neighboring nodes.

The modified Min-Sum decoding algorithm [9]–[11] is similar to the Sum-Product algorithm, with an approximation of check node process. It has some advantages in implementation over the Sum-Product algorithm, such as less computation complexity and no requirement of knowledge of SNR (signal-to-noise ratio) for AWGN channels. In the modified Min-Sum decoding algorithm, the check node processors compute the check-to-variable messages $R_{\rm cv}$ as follows:

$$R_{\rm cv} = \alpha \times \prod_{n \in N(c) \backslash v} \operatorname{sign}(L_{\rm vc}) \times \min_{n \in N(c) \backslash v} |L_{\rm vc}| \quad (1)$$

$$H = \begin{bmatrix} H_{11} & H_{12} & \cdots & H_{1t} \\ H_{21} & H_{22} & \cdots & H_{2t} \\ \cdots & \cdots & \ddots & \cdots \\ H_{c1} & H_{c2} & \cdots & H_{ct} \end{bmatrix}$$

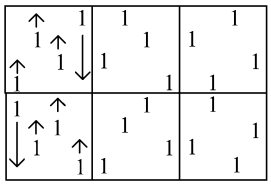


Fig. 1. Example of the (2, 3) shift-LDPC parity check matrix.

where α is a scaling factor around 0.75 [10], $L_{\rm vc}$ is the variable-to-check messages, $N(c)$ denotes the set of variable nodes that participate in cth check node.

The variable node processor computes the variable-to-check messages $L_{\rm vc}$ as the following:

$$L_{\rm vc} = \sum_{m \in M(v) \backslash c} R_{\rm mv} + I_v \tag{2}$$

where $M(v) \backslash c$ denotes the set of check nodes connected to the variable node v excluding the variable node c, I_v denotes the intrinsic message of variable node v.

B. Implementation Options

From the previously described decoding algorithm, it can be seen that the calculations are independent between different variable nodes or check nodes at each iteration. Therefore, LDPC codes are especially suitable for parallel implementation. In addition, the computation in the decoder is fairly simple, which means low logic consumption. The main obstacle for full parallelism is the complicated interconnection, which makes condensed chip layout impossible [3]. The problem can be alleviated through reducing the parallelism (partially parallel or serial processing) and using storage elements to store the intermediate messages passed along the edges of the graph [4]–[8]. Now the problem becomes how to reduce the complexity incurred by the message storage elements and how to speedup the decoding process.

In the following, we will introduce a new ensemble of LDPC codes, called *shift-LDPC codes*, to mitigate these decoder implementation problems.

III. Shift-LDPC Code Construction

A. Implementation-Oriented Code Construction

A regular $(N, M)(c, t)$ shift LDPC code example is shown in Fig. 1, with $N = t * P$ and $M = c * P$. As displayed, the parity check matrix consists of $c \times t$ submatrices. Each submatrix has a dimension of $P \times P$. The structures of the leftmost c submatrices are random column permutations of the identity matrix. The other submatrices are decided by the left-most c submatrices.

The "1"s in the shift LDPC code parity check matrix are arranged just as the example in Fig. 1. At first, the "1"s in the two leftmost submatrices are arranged randomly ensuring one "1" in each row and one "1" in each column. Then for each submatrix on the right-hand side, the "1"s are cyclic-shifted up by 1 space. The "1" at the top is moved down to the bottom. Finally, we can get a (c, t) regular shift-LDPC code.

B. Girth Optimization

The length of the shortest cycle in a Tanner graph is referred to as its girth. A large girth generally improves the bit error performance of the codes. Hence, LDPC codes with large girth are particularly desired.

Given a fixed block length, column weight, and row weight, the ensemble of shift-LDPC codes can have considerable variations in performance. To find the codes with large girth is an effective method to find good LDPC codes. Similar to the method used in [12], an efficient algorithm is developed for shift-LDPC codes to optimize the girth by exploiting the structured property of the codes. Normally, by using this girth optimization soft program, the cycle-4 can be easily eliminated, and the cycle-6 can be avoided for moderate code rate shift-LDPC codes.

C. Code Examples and Performance

Through extensive simulation, we found that the performance of the optimized shift LDPC codes can be comparable to the computer generated randomly codes.

Fig. 2 shows the performance comparison between a girth optimized (1008, 504) (3, 6) shift-LDPC code, the same size rate-1/2 code downloaded from Mackey's website and a (960, 480) irregular WiMAX code as well as comparison between a (8192, 7168) (4, 32)-regular shift-LDPC code and a girth optimized QC-LDPC code [12]. It can be seen that the WiMAX code performs better due to its irregular property, and in other cases the shift-LDPC codes have comparable performance. In fact, a portion of QC-LDPC codes can be converted to shift-LDPC codes with proper column permutation. On the other hand, the performance of shift-LDPC codes could be further improved by introducing limited irregularity, e.g., to zero out some of the submatrices, which will cause tiny hardware overhead.

IV. Decoder Architecture

In this section, we present the architecture specifically designed for shift-LDPC code decoders. For a regular $(N, M)(c, t)$ shift-LDPC code, P $(P = N/t = M/c)$ Variable node processor units (VPU) and M check node processor units (CPU) are instantiated in the decoder.

A. Decoding Schedule

First, let us discuss the decoding schedule. Fig. 3 illustrates the decoding flow for a simple shift-LDPC code. The schedule can be simply applied to other cases with different P, c, and t parameters.

P columns are processed concurrently in one clock cycle. The left-most P columns are processed first, then the second left-most P columns, and so on. The column process and the row process are interleaved. In every clock cycle, P VPUs get M check-to-variable messages and compute the M variable-to-check messages, so that M CPUs get one message each, so each CPU can deal with one step of the check node process. The

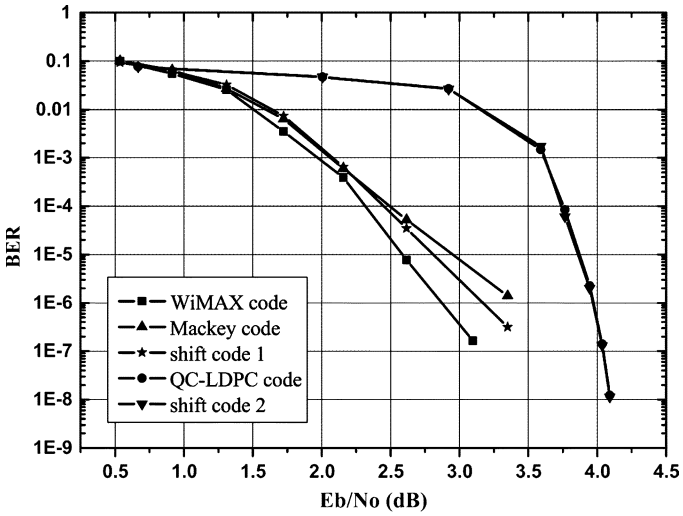


Fig. 2. Performance comparisons for (1008, 504) (3, 6) LDPC codes and (8192, 7168) (4, 32) LDPC codes.

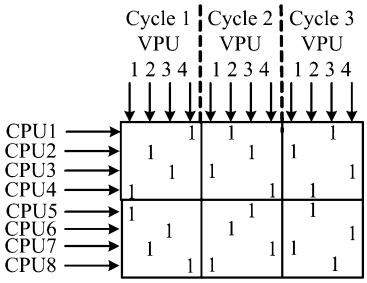


Fig. 3. Decoding flow for a sample code.

whole check node process is divided into t steps. With this decoding schedule, we can finish one iteration in t clock cycles. It is normally much faster than the traditional partly parallel decoder architectures [4], [5].

B. Overall Decoder Architecture

The overall decoder block diagram is shown in Fig. 4. The critical part of this implementation is the network connecting VPUs and CPUs. The shuffle network A transmits variable-to-check messages; the shuffle network B transmits check-to-variable messages. Shuffle network B is the same as shuffle network A except that the data flow is in the reverse direction. In the LDPC decoding algorithm, CPUs only communicate with VPUs and the row processes of different check nodes are independent of each other. In our shift-LDPC decoder architecture, one block called *CPU communication network* is added to the CPUs to reduce the complexity of the shuffle networks.

For random codes, the shuffle network routing complexity is normally intolerable, while for shift-LDPC codes we introduced above, the shuffle networks become really simple. Through the introduction of the *CPU communication network*, we can ensure that each CPU processes the variable-to-check messages from and transmit the check- to-variable messages to a fixed VPU during the entire decoding process. This will be illustrated more clearly later. In fact, the shuffle network connecting CPUs and VPUs only consists of M*b wires, where we assume each

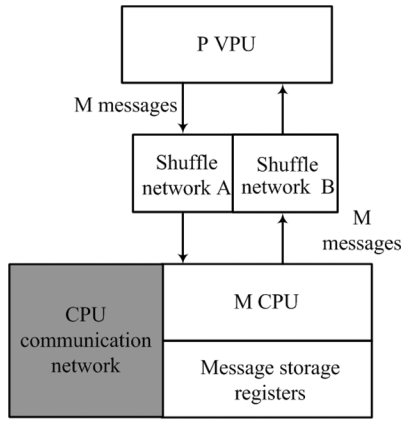


Fig. 4. Massages iteratively exchange between VPU and CPU through the shuffle network.

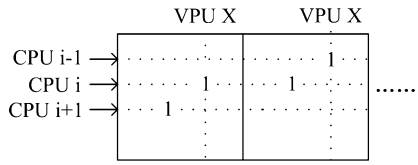


Fig. 5. Data flow of row process intermediate result.

message is quantized as b bits. Normally the quantization bits b is chosen as 6.

Fig. 5 explains why each CPU always computes with the messages from a fixed VPU during the entire decoding process. With the simple shuffle network, CPU i is connected with VPU X. The row process of each check node is separated into t steps with one variable-to-check message being processed in each step. At the first clock cycle of an iteration, having received the message from VPU X, CPU i performs the first step of ith row process. After that, CPU i passes the ith row process intermediate result to CPU $i+1$ through the *CPU communication network*. Meanwhile, CPU i receives the $(i-1)$th row process intermediate result from CPU $i-1$. In the second clock cycle of the iteration, CPU i still receives the variable-to-check message from VPU X. This message and the row process intermediate result in CPU i are both corresponding to row $i-1$. Therefore, CPU i can perform the second step of $(i-1)$th row process. At the same time, CPU $i+1$ is performing the second step of ith row process. Such process is continued until the whole iteration is finished.

For variable node processing, VPU X receives the check-to-variable message in sequential from row i to $i-1, i-2, \ldots$, etc. The *CPU communication network* can also ensure that VPU X only need to receive its message from CPU i.

We next show the above decoding schedule more clearly with Fig. 6. As an example, we will illustrate with the simple shift LDPC code matrix in Fig. 1. When one iteration starts, in the first clock cycle, each CPU processes the message from a VPU according to the "1"s positions in the left-most submatrix. When this one step row process is finished, the row process intermediate results are shifted between CPUs through the CPU communication network (intra iteration). In the second clock cycle, CPU2 process the data of row 1; CPU3 process

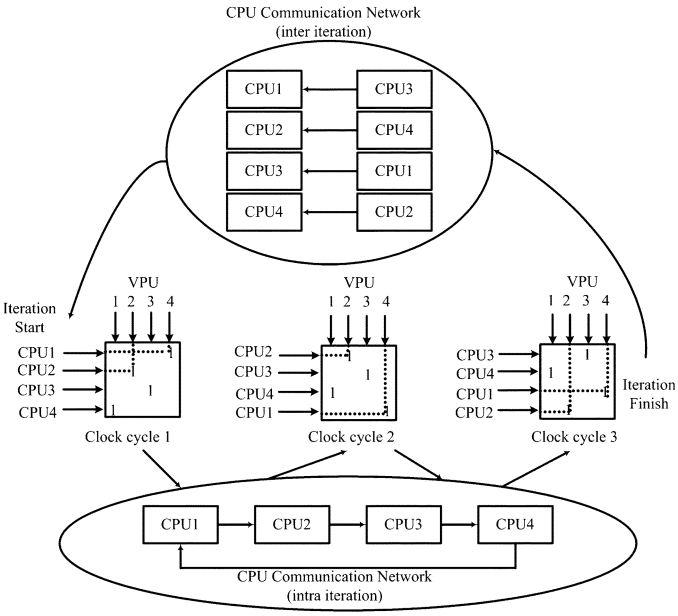


Fig. 6. Decoding schedule illustration.

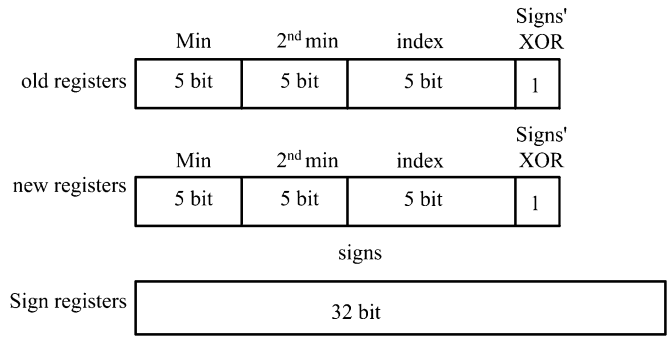


Fig. 7. Message memory management in CPU.

the data of row 2, etc. It can be seen from Fig. 6 that CPU1 always transmits to and gets message from VPU4; CPU2 always transmits to and gets message from VPU2; etc. In brief, each CPU communicates with the same VPU in all the three clock cycles per iteration. Once the iteration is finished, the row process result will be transferred back to appropriate CPU to produce check-to-variable messages. The results are transferred through the CPU communication network (inter iterations). This network is a one-to-one communication network.

Therefore, the communication between CPUs and VPUs required by the decoding algorithm is decomposed into three kinds of connections: the connection between CPU and VPU; the CPU communication network (intra iteration); and the CPU communication network (inter iterations).

In the following, we will introduce the architecture of check node processor applying the Min-Sum decoding algorithm. The (8192, 7168) (4, 32) shift LDPC code is chosen as the design example.

C. Architecture of Check Node Process Unit

First of all, we show the message memory management in Fig. 7. We only save the row process results. In addition, the results are saved in a compressed way that only the minimum magnitude, second-minimum magnitude, the location of the minimum and the sign bits are saved. This message storage method is similar to the method used in [8] and [9] and it can greatly reduce the memory requirement. These row process results are stored in three kinds of registers: old registers, new registers, and sign registers.

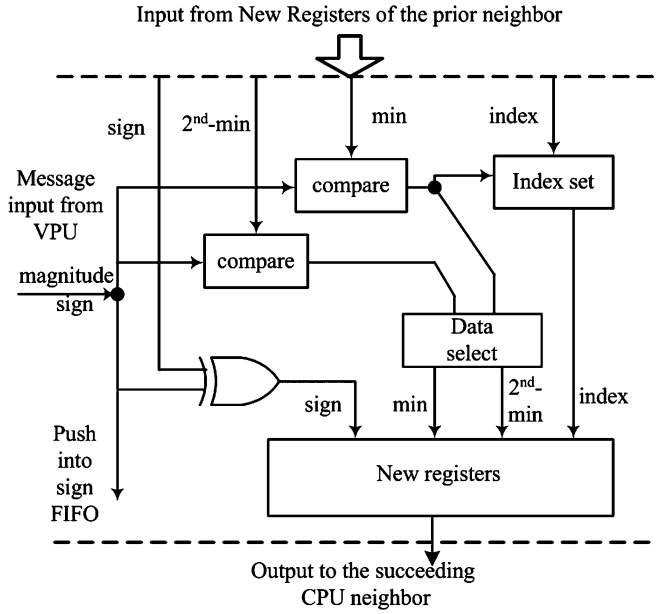


Fig. 8. Part 1 of the check node processor unit.

It is defined that the old registers contain the row process result of the last iteration; the new registers contain the row process intermediate result of the current iteration. The sign register group contains 32 sign bits of the variable-to-check messages. In each clock cycle, each sign register shifts in a sign bit of the variable-to-check message and shifts out a sign bit of a check-to-variable message. The data in new registers and old registers are always being passed between CPUs through the CPU communication network (intra iteration and inter iterations). The signs in the sign registers are for the same VPU, so they do not need to be passed.

Figs. 8 and 9 show the architecture of CPU. Because only one variable-to-check message is processed in one time, the computation in CPU has very low complexity. The CPU architecture is separated into three parts.

Part 1 is the row process computation part. The work it does is to compare the magnitude of the input message with the current row process intermediate result, and to update the sign and index. Then the updated row process intermediate result is registered and passed to its next CPU neighbor through the CPU communication network (intra iteration).

Part 2 is the check-to-variable message computation part. The work it does is to select the proper message magnitude according to the index value, and compute the sign of the message. Then the row process result of the last iteration is passed to next CPU. At the beginning of each iteration, it performs the message scale calculation ($\alpha = 0.75$).

Part 3 is a 32-bit FIFO. It receives a sign of input message from part 1 and pops out a sign to part 2 at each clock cycle.

TABLE I
SYNTHESIS RESULTS

Gate count	CPU combination logic	VPU combination logic	total combination logic	frequency (MHz)	Throughput per iteration (Gbps)	Message storage area (mm^2)	Total cell area (mm^2)
One parallel level design	148	953	395K	160	40.96	6.1	10
One parallel level+ pipelining	148	1410	512K	317	76.4 ★	6.1	11.3
Double parallel level	342	953	838K	151	77.3	6.1	14
Double parallel level + pipelining	342	1410	1072K	317	153	6.1	16.7

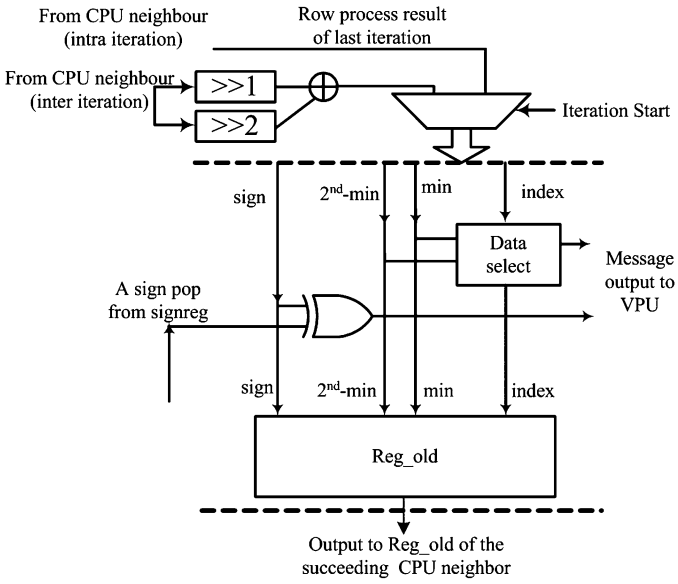


Fig. 9. Part 2 of the check node processor unit.

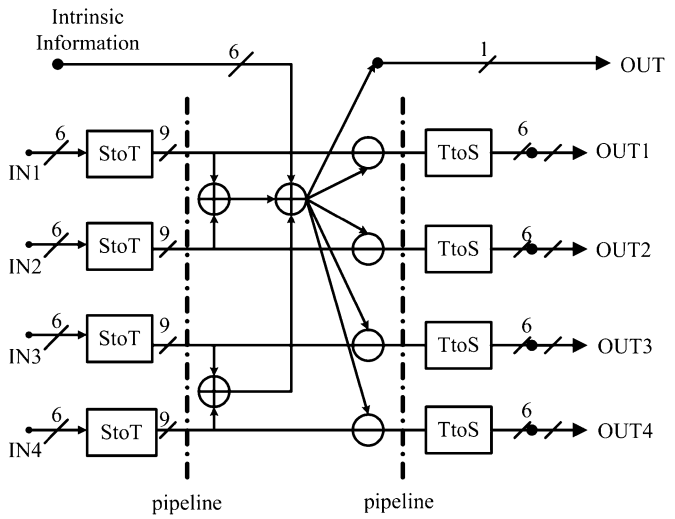


Fig. 10. Architecture of the variable node processor unit. StoT will convert from the sign-magnitude format to two's complement format. TtoS is for reverse conversion.

D. Architecture of Variable Node Processor Unit

The architecture of variable node processor unit is the same with [3], [4]. It can be designed with only combinational logic. To increase the clock frequency, two level pipelines are added to the VPU design. Fig. 10 shows the VPU architecture.

E. More Parallel Level

To further increase the decoding speed, more parallel level can be added to the decoder architecture. For example, to double the parallel level, 2*P VPUs should be instantiated. The architecture of CPU should be changed to process two messages per clock cycle. Then each iteration can be finished in t/2 clock cycles. A decoder example is designed to investigate the tradeoff between hardware cost and decoding speed.

V. IMPLEMENTATION OF SHIFT-LDPC CODE DECODERS

A. Synthesis Results

The technology used for this (8192, 7168) (4, 32) LDPC decoder implementation is a 0.18-μm CMOS process with 6 metal layers. Due to the large number of CPUs and VPUs (submatrix size $P = 256$), the bottom-up synthesis strategy is applied. Table I shows the synthesis result of four design examples: one-level parallel design, one-level parallel with pipelining, two-level parallel design, and two-level parallel with pipelining. It can be seen that, by applying pipelining, the clock frequency can be almost doubled, and the overhead is 81 registers per CPU. The one-level parallel design with pipelining can achieve the best tradeoff between the speed and the hardware complexity.

For the message storage, they are all implemented with registers. The transferring messages take $(16 * 2 + 32) * 1024 = 65\,536$ bits. In comparison, the traditional partially parallel architecture (e.g., [4]) needs to store $8192*4*6 = 196\,224$ bits in total. It saves 67% of the message memory needed. The initial intrinsic information takes $8192*6 = 49\,152$ bits. It is the same for both cases.

B. Floor-Plan and Layout

Fig. 11 shows the floor-plan and layout of the decoder chip with a die size of 4.1 mm $\times$ 4.1 mm. The logic density is 70%. The placement of CPUs is critical to reducing the routing congestion of CPU communication networks. The CPU array located in the center of the chip is specially arranged: the 1024 CPUs are aligned with 31 CPUs per row, so that the communications between CPUs can be locally routed.

VI. COMPARISON AND RESULTS

Table II shows the decoder implementation results compared with some other LDPC decoder architectures. The throughput we get is 5.1 Gb/s at a maximum iteration of 15. One parameter "Hardware Efficiency" as (Throughput $\times$ Iterations/Area) is defined to evaluate the efficiency of each architecture. The area metrics are scaled to resemble 90-nm CMOS results (65 nm by a factor of 2 and 180 nm by a factor of $1/4$). For the proposed design, the two values in area and "Hardware Efficiency" are

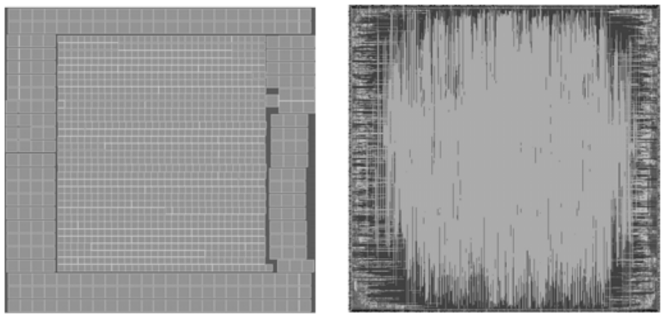


Fig. 11. Floor-plan and layout.

TABLE II
COMPARISONS WITH OTHER DECODER ARCHITECTURES

	Partly Parallel[5]	WiMax [13]	Ultra Sparse UWB[13]	Proposed
Code length	2048	576~2304	9600	8192
Code rate	1/2	1/2~5/6	3/4	7/8
Edges	6144	8448	26400	32768
Quantization	4b	6b	6b	6b
Algorithm	Min-Sum	3-Min	Min-Sum	Min-Sum
technology	180nm	65nm	65nm	180nm
frequency	125 MHz	400 MHz	500MHz	317 MHz
Iterations	10	20	10	15
Throughput	640Mb/s	399Mb/s	1.45Gb/s	5.1 Gb/s
Area (mm^2)	14.3	1.337	0.504	11.3 /16.8
Area [scaled to 90nm]	3.57	2.67	1.008	2.82 /4.2
Hardware Efficiency	1778	2989	14384	25500/ 18214

before/after place and route. The design results in [13] are after synthesis (do not include the layout utilization factors). Compared with the Ultra sparse code implementation proposed for UWB in [13], we can see that the proposed design can get more than 70% improvement in hardware efficiency while the higher clock speed benefited from much more advanced CMOS technology is not considered for this comparison. Otherwise our improvement will be even more significant.

From the implementation data shown in Table II, we can see that our decoder architecture is very efficient for high-speed LDPC decoder implementation.

REFERENCES

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

[2] D. J. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron Lett.*, vol. 32, pp. 1645–1646, 1996.

[3] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gbps 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.

[4] T. Zhang and K. Parhi, "A 54 Mbps (3, 6)-regular FPGA LDPC decoder," in *Proc. IEEE Sips'*, 2002, pp. 127–132.

[5] M. M. Mansour and N. R. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, Mar. 2006.

[6] Z. Wang and Z. Cui, "A memory efficient partially parallel decoder architecture for QC-LDPC codes," in *Conf. Signals, Syst. Comput., Record 39th Asilomar*, Oct. 2005, pp. 729–733.

[7] Z. Wang and Q. Jia, "Low complexity, high speed decoder architecture for Quasi-Cyclic LDPC codes," in *Proc. ISCAS*, 2005, pp. 5786–5789.

[8] M. Cocco, J. Dielissen, M. Heijligers, A. Hekstra, and J. Huisken, "A scalable architecture for LDPC decoding," *Proc. Des., Autom. Test Eur.*, vol. 3, pp. 88–93, Feb. 2004.

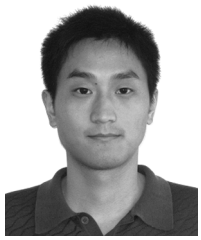
[9] F. Guilloud, E. Boutillon, and J. L. Danger, "λ-min decoding algorithm of regular and irregular LDPC codes," in *Proc. 3nd Int. Symp. Turbo Codes Related Topics*, Sep. 2003, pp. 451–454.

[10] J. Chen, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[11] J. Zhao, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes," *IEEE Trans. Commun.*, vol. 53, no. 4, pp. 549–554, Aug. 2005.

[12] J. Lu and J. M. F. Moura, "Partition-and-shift LDPC codes," *IEEE Trans. Magn.*, vol. 41, no. 10, pp. 2977–2979, Oct. 2005.

[13] T. Brack, "Low complexity LDPC code decoders for next generation standards," in *Proc. Des., Autom. Test Eur. (DATE)*, Apr. 2007, pp. 1–6.



Jin Sha received the B.S. degree in physics and the Ph.D. degree in microelectronics from the Nanjing University, Nanjing, China, in 2002 and 2007, respectively.

Since finishing his graduate study, he has been an ASIC Design Engineer with the Corporate Research and Development Group, OmniVision Technologies, Inc. His research interests include VLSI architectures and integrated circuit (IC) design for communications, coding theory applications, and image signal processing.

Zhongfeng Wang (M'00–SM'05) received the B.E. and M.S. degrees in automation from Tsinghua University, Beijing, China, in 1988 and 1990, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Minnesota, Minneapolis, in 2000.

Upon finishing his graduate study, he worked for Morphics Technology Incorporated for two years and then moved to National Semiconductor Company in 2002. From 2003 to 2007, he worked for the School of Electrical Engineering and Computer Science, Oregon State University as an Assistant Professor. He is now a Senior Principle Scientist with Broadcom Corporation, CA. He has published numerous technical papers and has filed several U.S. patent applications. His current research interests include the area of VLSI design for digital communication systems.

Dr. Wang was a recipient of the Best Student Paper Award at the IEEE Workshop on Signal Processing Systems (SiPS'99) in 1999 and corecipient of the IEEE Circuits and Systems Society VLSI Transactions Best Paper Award in 2007. He has served as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS and is serving as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS. He has also served in various technical committees for IEEE and ACM conferences. He is a member of Sigma Xi.

Minglun Gao received the B.S. degree in electronics from Tsinghua University, Beijing, China, in 1968, the M.S. degree in the electrical engineering from Hefei University of Technology, Hefei, Anhui, China, in 1981, and the Ph.D. degree in electrical engineering from University of Dayton, Dayton, OH, in 1991.

Before 1997, he worked for an electronics company (China) and Philips Semiconductors (USA), respectively. He is currently a Professor with the Department of Physics, Nanjing University, Nanjing, Jiangsu, China, and a Professor with the Electrical Engineering Department, Hefei University of Technology. He is currently a Visiting Professor with Shandong University, Jinan, Shandong, China, since 2003. He is also the Chief Scientist of the National IC Design Industrialization Base (Wuxi, China). His research interests include SoC-IP design methodology, MP-SoC architecture, NoC architecture, and hardware acceleration for EDA.

Prof. Gao is a cochairman of the National Standard Group for SIP (China) since 2002 and has been the chair for annual conferences all of the years.

Li Li received the B.E. degree in electric engineering and automation and the Ph.D. degree in precision instrument from Hefei University of Technology, Hefei, China, in 1996 and 2002, respectively.

She worked for the Physics Department, Nanjing University, Nanjing, Jiangsu, China, after finishing her graduate study and is currently an Associate Professor. She has published numerous technical papers and coauthored two books. She has filed for three China invention patents. Her current research interests include the area of VLSI design for digital communication systems and multi-processor system-on-a-chip (MPSoC) architecture design.

Dr. Li was a recipient of the Excellent Faculty of Jiangsu Six Talent Peaks Project in 2004 and the Excellent Young Faculty of Nanjing University in 2006.