Proceedings of the 5th WSEAS Int. Conf. on Instrumentation, Measurement, Circuits and Systems, Hangzhou, China, April 16-18, 2006 (pp224-229)

High-Throughput and Memory Efficient LDPC Decoder Architecture

JIN SHA, MINGLUN GAO, ZHONGJIN ZHANG,LI LI

Institute of VLSI design
Key Laboratory of Advanced Photonic and Electronic Materials
Nanjing University
Nanjing, China, 210093

ZHONGFENG WANG

School of EECS
Oregon State University
Corvallis, OR 97331-5501,USA

Abstract: Low-Density Parity-Check (LDPC) code is one kind of prominent error correcting codes (ECC) being considered in next generation industry standards. The decoder implementation complexity has been the bottleneck of its application. This paper presents a new kind of high-throughput and memory efficient LDPC decoder architecture. In general, more than fifty percent of memory can be saved over conventional partially parallel decoder architectures. It is shown that this presented hardware structure will be highly competent in high throughput and low decoding latency applications.

Key-Words: - Low-density parity-check (LDPC) codes, VLSI architecture, decoder, shift LDPC

1. Introduction

Error Correction Codes (ECC) is widely applied in modern digital communication systems. Turbo codes and LDPC codes [1] are the two most popular ECC near the Shannon limit. Turbo code is overwhelmed by LDPC code in some aspects such as lower error floor, less computation requirement and fully parallel decoding schemes.

However, the implementation of LDPC decoder is not trivial. To implement it directly in its inherent parallel manner may get the highest decoding throughput. But for large codeword length (bigger than 1K), to avoid routing conflict, the complex interconnection will take up more than half of the chip area [2]. Serial VLSI architecture or partly parallel architecture is well studied nowadays [3] [4] [5] [6], which uses RAM to store messages transferred between variable nodes (VN) and check nodes (CN). Therefore, the huge memory requirement for message storage will be a problem. In addition, the low throughput or high decoding latency remains the major shortcoming of the serial implementation.

As for the decoding algorithm, the minimum-sum algorithm [7] [8] is an approximation of the Sum-Product algorithm. From the perspective of implementation, the Min-Sum algorithm requires less computation and estimation of noise power is unnecessary for an additive white Gaussian noise (AWGN) channel. Furthermore, the Min-Sum algorithm can help reducing the message storage requirement because the messages transmitted from a check node to adjacent variable nodes per iteration have only two possible magnitudes. However, this advantage is not easy to take in hardware implementation. In [5], the authors implemented an LDPC decoder utilizing the Min-Sum algorithm. The memory requirement reduction is achieved, but the hardware is complex, and the decoding latency is extremely high. The fully efficient use of Min-Sum algorithm in decoding LDPC codes still remains unresolved.

This paper proposes a high-throughput and memory efficient LDPC decoder architecture. In Section 2, the code construction suited for this kind of architecture is briefly introduced. The decoder architecture is presented in Section 3. Section 4 gives some results, and Section 5 concludes the paper.

2. Code Construction and Decoding Algorithm

LDPC codes can typically be defined by an $M \times N$ parity check matrix **H**. The symbol N, represents

the length of the block (i.e. the number of bits in the codeword), while the symbol *M*, represents the number of parity checks in the code. The rate of such a code is thus *(N-M)/N*. In addition, the LDPC code is defined as regular if each row has the same number of '1' and each column also has the same number of '1'. The matrix of a regular *(N,M) (c,t)* shift LDPC code *(N=t*P, M=c*P)* is shown in Fig.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. Parity check matrix of shift LDPC codes

As displayed in Fig.1, the parity check matrix consists of $c \times t$ sub-blocks, each with a *P* row and *P* column matrix. Each sub-block is called a cell. In each cell, there are *P* '1's, one in each row, and one in each column.

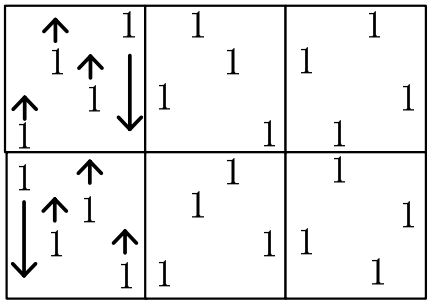


Fig.2. An example of the (2,3) parity check matrix

The '1's of sub-matrices are arranged just as the example in Fig.2. At first, the '1's in the most left cell are arranged randomly. Then for every cell on the right side, the '1's are moved up by 1 space. The '1' at the top is moved down to the bottom.

Through extensive simulation, we find the performance of shift LDPC code can be comparable to the randomly generated codes. In addition, the Quasi-Cyclic LDPC code is a subset of shift LDPC code. It can be converted to shift LDPC form and be applied to the architecture described below.

The typical LDPC decoding algorithm is the sum-product algorithm which has two phases. In the first phase, the variable nodes compute updated information which is sent to adjacent check nodes. In the second phase, the check nodes compute updated information based on the new massages from the variable nodes. This update information is then sent back to adjacent variable nodes and the process is repeated.

The modified min-sum decoding algorithm is similar to the sum-product algorithm, with an approximation of check node process.

In the modified min-sum decoding algorithm, the check node processors compute the check-to-variable messages Rcv as the following:

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

where α is a scaling factor. L_{cv} is the variable-to-check messages. *N(c)* denotes the set of variable nodes that participate in cth check node.

The variable processor computes the variable-to-check messages L_{cv} as the following:

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

M(v)\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*.

3. Decoder Architecture

The novel decoder architecture corresponding for shift LDPC code is scaled parallel decoder architecture. *P* Variable node Processor Units (VPU) and *M* Check node Processor Units (CPU) are instantiated in the decoder.

Here the decoding schedule is firstly presented. Fig.3 illustrates the decoding schedule for a simple (12,8) (2,3) parity check matrix. The schedule is in the same way for different *P, c* and *t* parameters.

P columns are processed concurrently in one clock cycle. The most left *P* columns are processed first, then the right *P* columns, and so on. In every clock cycle, *P* VPUs get *M* check to variable messages and compute the *M* variable to check message, so that *M* CPUs get one input each, so every CPU can deal with one step of the check node process. The whole check node process is divided into *t* steps. With this decoding schedule, it can finish per iteration in *t* clock cycles. It is normally much faster than the traditional partly parallel decoder architectures [3]).

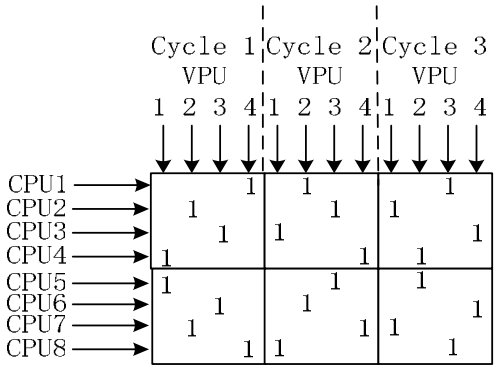


Fig.3. Decoding schedule

3.1 Overall decoder architecture

The overall decoder architecture is shown in Fig.4. The critical part of this implementation is the shuffle network. For random code, the routing complexity will be intolerable, while for shift LDPC codes introduced above, the shuffle network become really simple. The message registers in CPUs are connected together as circled shift registers. In every clock cycle, each CPU does the computation and gives the result to its next CPU neighbor so that every CPU can process the message from the same VPU all the time. This will be illustrated more clearly later. Then the shuffle network can be really simple *M*b* wires (*b* is the quantized message bits). The transfer of *M* returning messages from CPUs to VPUs is similar to the messages from VPUs to CPUs.

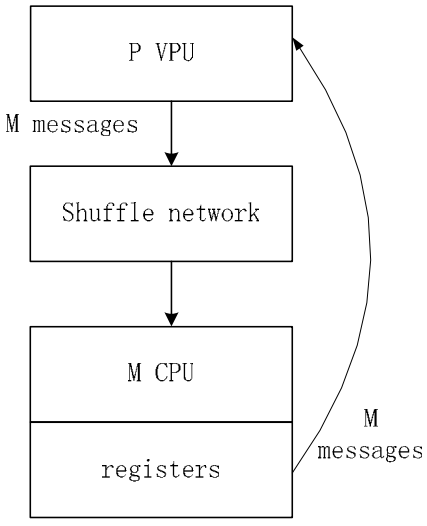


Fig.4. Massages keep exchanging between VPU and CPU through the shuffle network

Fig.5 explains why CPU i always compute with the messages from VPU X during the whole decoding process. By the simple shuffle network, CPU i is connected with VPU X. At the first clock cycle of one iteration, having received the message from VPU X, CPU i performs one step of the check node process, and stores the value corresponding to row i in CPU i+1. Meanwhile, CPU i receives the result of CPU i-1, which is the current value corresponding to row i-1. In the next cycle, CPU i still receives message from VPU X, so the input message and the intermediate result in CPU i are both for row i-1, now it can continue to perform next step of the check process of row i-1. And, CPU i+1 is performing the check process of row i. Continue such process cell by cell until the whole iteration is finished.

For variable node process, VPU X should receive input message in sequence from row i, i-1, i-2…. So the registers storing result of the last iteration are also shifted to ease the connection between VPU and these registers.

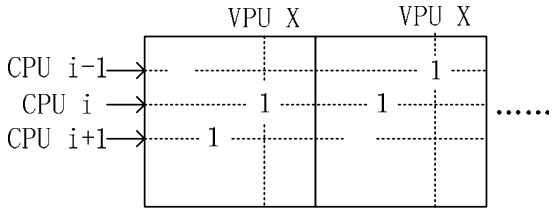


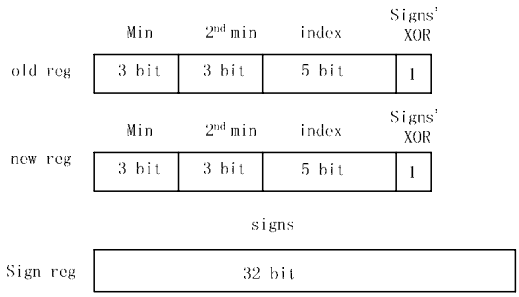
Fig.5. shift the result of check node process

In the following, the architecture of check node processor applying the Min-Sum algorithm will be introduced. As an example, a (8192, 7168) (4, 32) regular LDPC code similar to a code mentioned in [9] is chosen.

3.2 Architecture of Check Processor Unit

First of all, Fig.6 shows the memory management. There are three groups of registers: old registers, new registers, and sign registers. This is similar to the method used in [4] in that this method can greatly reduce the message memory requirement. The old register group contains the result of the last iteration. The new register group contains the values of the current iteration. They are both being shifted all the time to ensure that every CPU is receiving message from and delivering to the same VPU. After each iteration, the new reg i contains the min, 2^{nd}-min, index and the xor of all the signs of row i-31(mod P), as explained above. These values are transferred to the old register group to provide messages from check nodes to variable nodes in the next iteration. The sign register group contains 32 message signs of the

variable to check messages, and they are not shifting between each other. In every 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 (has to do xor with the sign bit in current old reg to gain it). Remember that they are always for the same VPU, so they do not need to be shifted between each other.



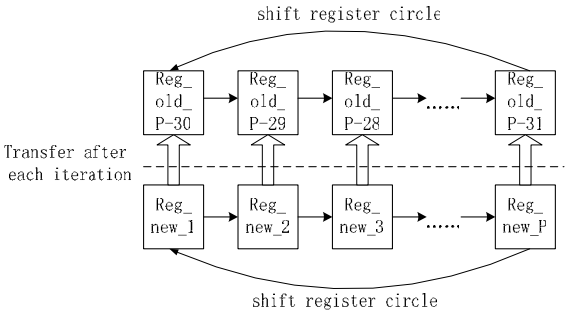


Fig.6. Register management in CPUs

Fig.7~9 shows the architecture of CPU. Because its input consists of only one message, the computation in this CPU is of extremely low complexity compared to other architectures.

Part 1 is the check computation part. The work it does is to compare the magnitude of input with the minimum and 2nd minimum value from the last neighbor, and to update the sign and index. Then the updated results are saved in reg_new to transfer to its next neighbor.

Part 2 is the check to variable message output part. The work it does is to select the proper message magnitude according to the index value, and compute the sign of the message by xor of the sign bit in reg_old and the bit popped out from part3. Then the updated results are stored in reg_old to transfer to its next neighbor.

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

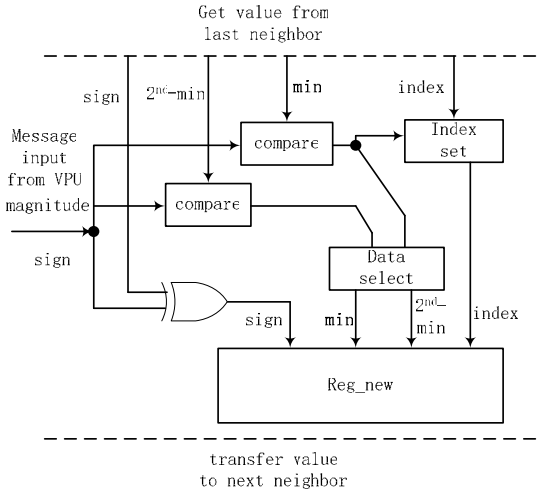


FIG.7. Part 1 of the check node processor

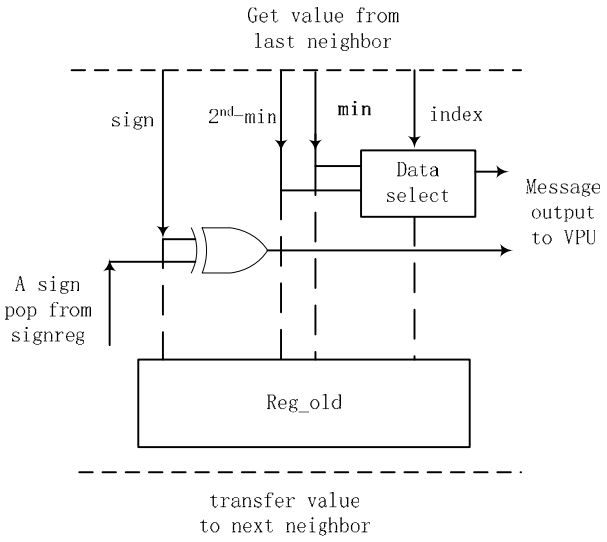


FIG.8. Part 2 of the check node processor

FIG.9. Part 3 of the check node processor

3.3 Architecture of VPU

The architecture of our variable processor unit is the same with others [2] [3]. An extra small memory is embedded in every VPU to store the channel information. Fig.10 shows the architecture.

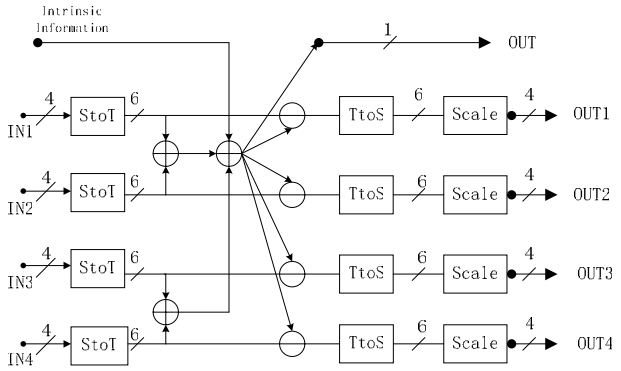


FIG.10. Architecture of the variable node processor

4. Results

Here gives some analysis on the decoding speed and the throughput of the proposed architecture. For a code with codeword length *N*, maximum iteration times *I*, clock frequency *f*, a maximum throughput of *f*N/(t*I)* can be achieved, *1/t* of the fully parallel decoder in math. The decoding latency is *(t*I/f)*. Due to the significantly reduced data path length in CPU and the simplified interconnection complexity, the proposed decoder should be able to run with a much faster clock. In addition, with the large N it can bear, very high throughput and small decoding latency can be achieved. Compared with a fully parallel architecture, our decoder keeps the excellence in throughput and decoding delay, and totally eliminates the routing problem for large code length. Note that the wires between shift registers are locally routed and will not induce routing problems.

In addition, because of the efficient use of Min-Sum decoding algorithm, the memory requirement is greatly reduced in this architecture. For the (4, 32) regular code mentioned above, assuming a four bits quantization method, the traditional architecture will need to store 8176*4*4 = 130816 bits in total for the exchanging messages. Applying the architecture presented in this paper, it only need to store (12*2+32)*1024 = 57344 bits in total. It saves more than half of the message memory needed. And for a more precise quantization method, such as six bits quantization, more memories can be saved. Note that the message memory usually occupies more than 70% of the decoder area.

More recently, a decoder for the (8192, 7168) (4, 32) regular QC-LDPC code is implemented. The technology that has been used for implementation is a 0.18um CMOS process with 6 metal layers. The area is 4mm×4mm with 70% logic utilization. The maximum clock frequency of the decoder is 200MHz and decoder achieves a throughput of 2.4Gbit/sec (with 20 iteration times). As a comparison with other high throughput LDPC decoders, in [2], a 1024-bit irregular LDPC decoder with 4-bit message passing is described which has a die size of 52.5mm^2 and operates at 64MHz and has a throughput of 1 Gbit/sec (with 64 iteration times).

Fig.11 shows the floorplan of the decoder. The CPU array located in the center of the chip is specially planed. The 1024 CPUs are aligned with 31 CPUs per row. Hence, most of the wires connecting between new_regs, between old_regs and between old_reg and new_reg are locally routed.

In a word, the proposed decoder architecture are very competent in dealing with hardware complexity, throughput and decoding latency.

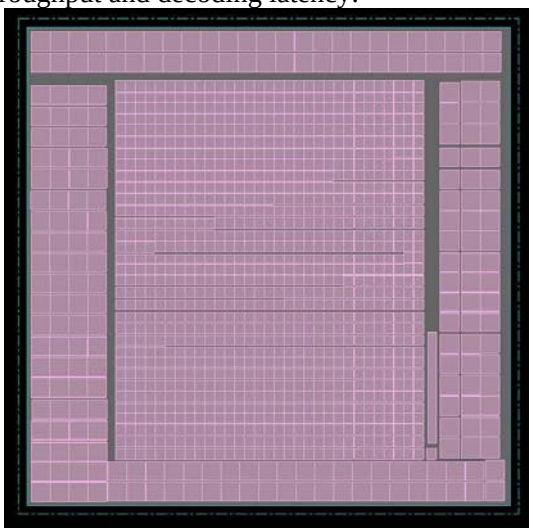


FIG.11. Floorplan of the 8192-bit LDPC decoder

5. Conclusions

As many research groups have discovered in the last couple years, several different VLSI architectures can be applied to the design of LDPC decoders. This paper has presented a new kind of high-throughput, memory efficient LDPC decoder architecture. The code ensemble which is compatible with this kind of architecture, called Shift-LDPC, has insignificant

error-correcting performance degradation compared with the codes randomly generated. The key is to save the massage memory requirement by efficiently applying the min-sum decoding algorithm and to reduce the routing complexity by adding some local communications between check node processors. By applying this novel decoder architecture, more than fifty percent of memory can be saved and throughput in excess of 2Gbps can be achieved. This proposed hardware structure will be highly competent in high throughput, low decoding latency applications.

6. Acknowledgments

The work presented in this paper was supported by the Foundation of High-Tech of Jiangsu Province of China under Grant No.BG2005030; the National Nature Science Foundation of China under Grant No.90307011.

References:

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

[2] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gbps 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder," IEEE Journal of Solid-State Circuits, vol. 37, pp. 404-412, 2002

[3] T. Zhang, "Efficient VLSI Architectures for Error-Correcting Coding," Ph.D. Thesis, Univ. of Minnesota, 2002.

[4] M. Cocco, J. Dielissen, M. Heijligers, A. Hekstra, and H. Huisken, "A Scalable Architecture for LDPC Decoding," in Proc. of the Design, Automation and Test in Europe Conference and Exhibition Designers' Forum (DATE'04). vol. 3, pp. 88-93, Feb. 2004.

[5] M. M. Mansour and N. R. Shanbhag, "Low-Power VLSI Decoder Architectures for LDPC Codes," in Proc. ISLPED'02, pp. 284-289, 2002.

[6] Z. Wang and Q. Jia, "Low Complexity, High Speed Decoder Architecture for Quasi-Cyclic LDPC Codes," in Proc. ISCAS'05, pp. 5786-5789, May 2005.

[7] J. Heo, "Analysis of Scaling Soft Information on Low Density Parity Check Codes," Elect. Letters, vol. 39, pp. 219–221, Jan. 2003.

[8] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and H. Xiao-Yu, "Reduced-Complexity Decoding of LDPC Codes," IEEE Transactions on Communications, vol. 53, pp. 1288-1299, Aug. 2005.

[9] L. Chen, J. Xu, I. Djurdjevic and S. Lin, "Near-Shannon-Limit Quasi-Cyclic Low-Density Parity-Check Codes," IEEE Transactions on Communications, vol. 52, pp. 1038-1042, Jul. 2004.

[10] F. Guilloud, "Generic Architecture for LDPC codes Decoding," Ph.D. Thesis. 2004. Http://pastel.paristech.org/archive/00000806/01/these.pdf.