TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol. 12, No. 11, November 2014, pp. 7721 ~ 7727
DOI: 10.11591/telkomnika.v12i11.6512

A New Low-Costing QC-LDPC Decoder for FPGA

Zhao Han[1], M.R. Anjum*[2]
[1]School of Information Science and Technology, Jinan University, Guangzhou, Guangdong Province, 510632, China
[2]School of Information & Electronics, Beijing Institute of Technology, Beijing, China. 100081.
*Corresponding author, e-mail: 328623033@qq.com

Abstract

Based on the Generalized Distributive Law and the features of FPGAs, this paper proposes a new strategy for implementation of Low-Costing QC-LDPC Decoder on FPGA platform. We get this new strategy from the Generalized Distributive Law, which is proposed to describe the belief propagation on graphs. And using this new strategy a low-costing (2560, 1024) LDPC decoder is implemented on a Xilinx Virtex-4 FPGA. Results show that this new strategy can make good use of the performance of LDPC codes, even though it needs less resource.

Keywords: *LDPC decoder, low-costing, generalized distributive law, FPGA*

1. Introduction

Low-Density Parity-Check (LDPC) Codes were first proposed by R.G. Gallager [1] in 1963 and rediscovered by David J.C. Mackay [2] in 1996. LDPC Codes has been attracting a large amount of attention since then because of the good error-correcting performance and parallel decoding. Moreover, many standards have adopted LDPC codes as channel coding, such as IEEEStd802.16, CCSDS 131.1-O-2 and etc.

In practical implementations of many error-control and signal processing systems, field programmable gate arrays (FPGAs) are widely used because it carries many advantages. As the design of high throughput and low-costing LDPC decoder is in great need, more and more LDPC decoders constructed on FPGAs are designed. Reference [3] proposed implementation of a low complexity soft-input soft output (SISO) LDPC decoder. With (1008,504) LDPC codes, the costs of the LDPC decoder are 1109 logic elements and 210,944 RAM bits. In[4], a flexible partially-parallel LDPC Decoder is proposed, and it achieved 50Mbps throughput with the use of 2,778 slices and 29 Block RAMs on a Virtex-2 FPGA. Moreover, a structure for LDPC decoder for long code length and is proposed in [5], in which implements a (9536, 4768) decoder with the use of 34127 logic elements and 102RAMs. So designing a FPGA based, low-costing and practical LDPC decoder is necessary.

In this paper, a parallel and low-costing LDPC decoder architecture for FPGA is proposed, as well as an optimization of the memory system is presented. The architecture is implemented on a Xilinx Virtex-4 FPGA, (2560,1024) LDPC code is chosen. The rest of the paper is organized as follow. In the second section, the new strategy of LDPC decoder is being proposed from the Generalized Distributive Laws (GDL) [6]. In section three, a (2560, 1024) LDPC decoder is implemented using the new decoding strategy. And the results and analysis are presented in section four.

2. LDPC Decoding Algorithms

When Gallager first proposed LDPC codes in 1962, a probabilistic decoding method was also presented, and the Belief-Propagation Algorithm was introduced to LDPC decoding by Mackay. As we know, the BP Algorithm can be viewed as massage passing on graphs, and it obeys the Sum-Product Updating Laws (SPUL) [7] or the GDL.

Received July 21, 2014; Revised August 24, 2014; Accepted September 10, 2014

2.1. The LDPC Decoding in Factor Graphs

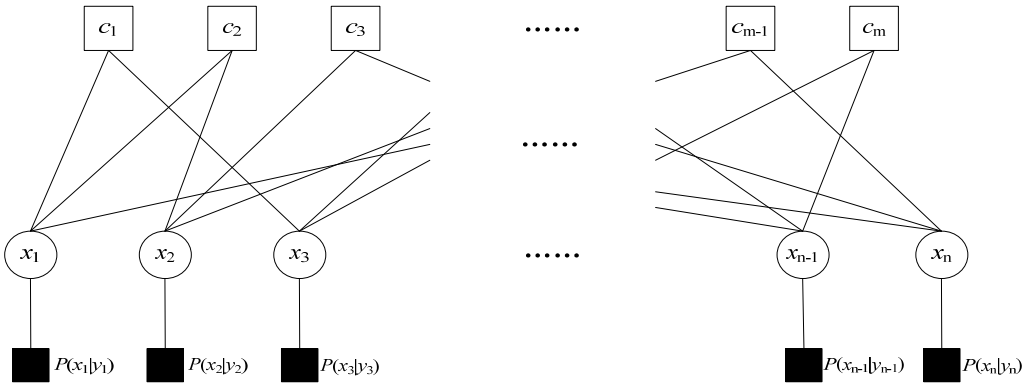

Figure 1. The Function of LDPC Described by Factor Graphs

Under the definition of Factor Graphs, the function of a LDPC code described by a m×n Matrix can be defined as the graph in Figure 1. So, based on the SPUL, the decoding process can be described as the BP algorithm, and if we define these passing massages in Log-Likelihood-Ratio(LLR) and do some modifications in its check node update step, it becomes the Min-Sum algorithm (MSA). Both of the two algorithms are well known, and will not be stated carefully in this paper again.

2.2. The LDPC Decoding in GDL

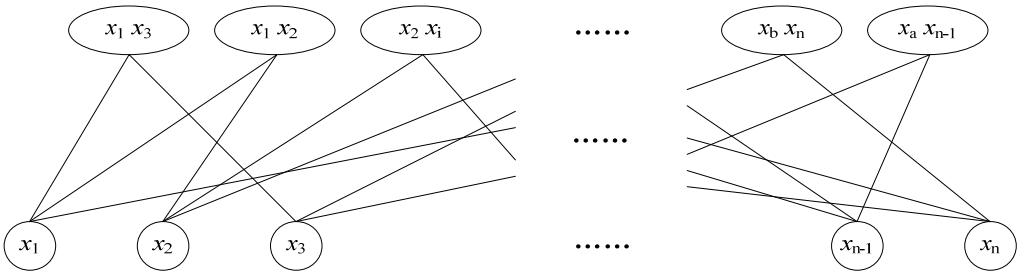

Figure 2. The Function of LDPC Described by GDL

Table 1. The Definition of Local Domain and Local Kernel Figure 2

local domain	local kernel
$\{x_1\}$	$P(x_1\|y_1)$
$\{x_2\}$	$P(x_2\|y_2)$
…	…
$\{x_n\}$	$P(x_n\|y_n)$
$\{x_1, x_3\}$	1
$\{x_1, x_2\}$	1
…	…
$\{x_a,x_n\}$	1

Moreover, based on the definition of the Generalized Distributive Laws (GDL), we can define the local domain and the local kernel as Table 1, and the function of a LDPC code can be described as Figure 2. Hence, the decoding process can be described as follow:

(1) Initialization

$$r_{ji}^{(l)}(0) = 1 \tag{1}$$

$$r_{ji}^{(l)}(1)=1 \tag{2}$$

(2) Variable-node update

$$q_{ij}^{(l)}(0)=K_{ij}P(\mathrm{c}_i=0\,|\,\mathrm{y}_i)\prod_{j'\in C_i\setminus j}r_{j'i}^{(l-1)}(0) \tag{3}$$

$$q_{ij}^{(l)}(1)=K_{ij}P(\mathrm{c}_i=1\,|\,\mathrm{y}_i)\prod_{j'\in C_i\setminus j}r_{j'i}^{(l-1)}(1) \tag{4}$$

Where $q_{ij}^{(l)}$ is the massage passing from *ith* variable node to *jth* check node in *lth* iteration, C_i is the set of variable nodes that adjusted to the *ith* variable node, and $C_i \setminus j$ is C_i without the *jth* check node. Moreover, K_{ij} is the normalization constant which is $q_{ij}^{(l)}(1)+q_{ij}^{(l)}(0)$.

(3) Check-node update

$$r_{ji}^{(l)}(0)=\frac{1}{2}+\frac{1}{2}\prod_{i'\in R_j\setminus i}\left(1-2q_{i'j}^{(l-1)}(1)\right) \tag{5}$$

$$r_{ji}^{(l)}(1)=\frac{1}{2}-\frac{1}{2}\prod_{i'\in R_j\setminus i}\left(1-2q_{i'j}^{(l-1)}(1)\right) \tag{6}$$

Where $r_{ji}^{(l)}$ is the massage passing from the *jth* check node to the *ith* variable node in *lth* iteration, R_j is the set of variable nodes that adjusted to the *jth* check node, and $R_j \setminus i$ is R_j without the *ith* variable node.

(4) Computing the Pseudo-posterior Probability

$$Q_i^{(l)}(0)=K_{ij}P(\mathrm{c}_i=0\,|\,\mathrm{y}_i)\prod_{j'\in C_j}r_{j'i}^{(l-1)}(0) \tag{7}$$

$$Q_i^{(l)}(1)=K_{ij}P(\mathrm{c}_i=1\,|\,\mathrm{y}_i)\prod_{j'\in C_j}r_{j'i}^{(l-1)}(1) \tag{8}$$

Where $Q_i^{(l)}$ is the pseudo-posterior probability of *ith* bits after the *lth* iteration, and K_i is the normalization constant which is $Q_i^{(l)}(1)+Q_i^{(l)}(0)$.

(5) Decoding

If $Q_i^{(l)}(1)>Q_i^{(l)}(0)$, the *ith* bit will be decoded as 1, else the *ith* bit will be decoded as 0. And if the decoded code word satisfies $\hat{C}H^T=0$ or the decoding process reaches the max iterations we set, the decoding process stops.

Similar with the convert from BP Algorithm to MSA, we can also describe the steps of LDPC decoding as follow:

(1) Initialization

$$L(r)_{ij}^{(0)}=0 \tag{9}$$

(2) Variable-node update

$$L(q)_{ij}^{(l)} = L(P)_i + \sum_{j' \in C_i \setminus j} L(r)_{j'i}^{(l)} \quad (10)$$

Where $L(q)_{ij}^{(l)} = \ln \frac{q_{ij}^{(l)}(0)}{q_{ij}^{(l)}(1)}$ is the LLR massage passing from the *ith* variable node to the *jth* check node in *lth* iteration.

(3) Check-node update

$$L(r)_{ji}^{(l)} = \prod_{i' \in R_j \setminus i} \operatorname{sgn}\left(L(q)_{i'j}^{(l)}\right) \min_{i' \in R_j \setminus i}\left(\left|L(q)_{i'j}^{(l)}\right|\right) \quad (11)$$

Where $L(r)_{ij}^{(l)} = \ln \frac{r_{ij}^{(l)}(0)}{r_{ij}^{(l)}(1)}$ is the LLR massage passing from the *jth* check node to the *ith* variable node in *lth* iteration.

(4) Computing the Pseudo-posterior Probability

$$L(Q)_{ij}^{(l)} = L(P)_i + \sum_{j' \in C_i} L(r)_{j'i}^{(l)} \quad (12)$$

Where $L(Q)_i^{(l)} = \ln \frac{Q_i^{(l)}(0)}{Q_i^{(l)}(1)}$ is the LLR pseudo-posterior probability of *ith* bits after the *lth* iteration.

(5) Decoding

If $L(Q)_i^{(l)} > 0$, the *ith* bit will be decoded as 0, else the *ith* bit will be decoded as 1. And if the decoded code word satisfies $\hat{C}H^T = 0$ or the decoding process reaches the max iterations we set, the decoding process stops.

According to the features of FPGAs, we adopt the second decoding method to design the LDPC decoder we proposed. Because the second decoding method can initialize the decoder by assign 0 to all $L(r)_{ji}$, and this step can be realized in the initialization of FPGA at once, while the first one need extra assignment operation to assign the value of the received bits to $L(q)_{ij}$ the value of the received bits. Hence, the second decoding method uses fewer sources than the first one. In fact, in the first iteration, the variable-nodes update step finishes is equal to the Initialization in MSA, but this "Initialization" use the variable-nodes update instead of adding extra assignment operation to the decoder.

3. Implementation

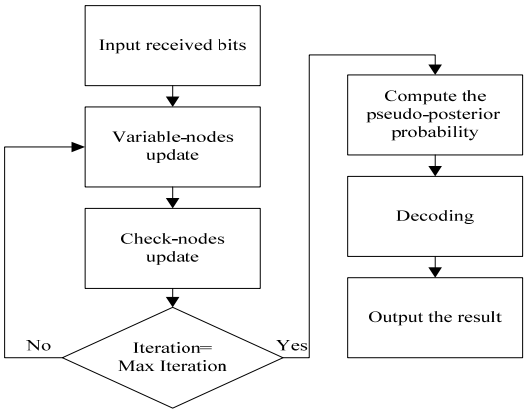

Figure 3. The Procedure of (2560, 1024) QC-LDPC Decoder

Using the decoding algorithm we proposed above, a (2560, 1024) QC-LDPC decoder on FPGA is implemented in this section. The procedure of the decoder is shown in Figure 3, to further reduce the resources consumption, we ignore the computation of the syndrome, and set a max iteration through the Monte Carlo simulation method [8].

The structure of the decoder is shown in Figure 4, the Solid lines stand for the direction of data passing, while the dashed lines stand for the direction of control signal and addressing signal. The RAM_p, RAM_r and RAM_q represent RAMs that store the values of the received bits, the value of $L(r)_{ji}$ and the value of $L(q)_{ij}$ respectively. Two address generators generate the read/write addresses of RAM_p, RAM_r and RAM_q. The variable-nodes update step and the check-nodes update step will be preceded in the variable_nods_update model and the check_nods_updatemodel, and the decoding model decides which digit the input bit should be, 1 or 0. Moreover, the processing control model generates control signal for address generator, nodes update model decoding model according to procedure of the decoder.

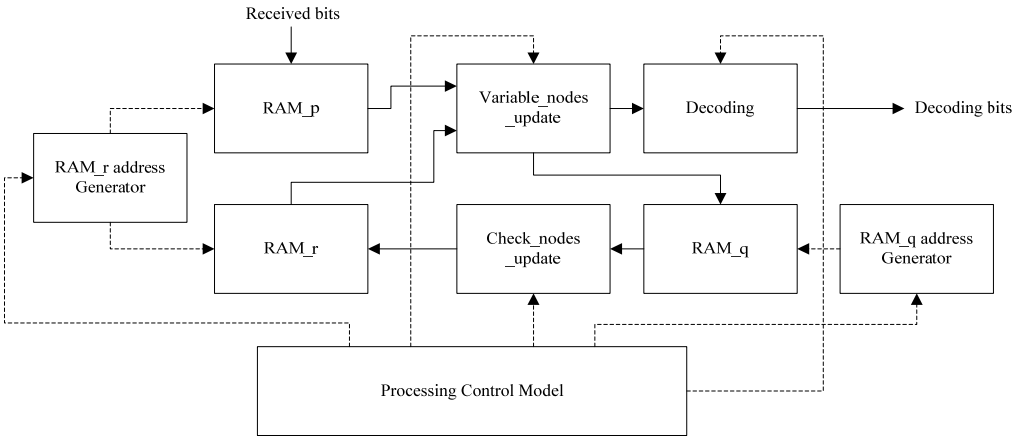

Figure 4. The Structure of (2560, 1024) QC-LDPC Decoder

As the check matrices of LDPC codes are always very large, the number of massage passed between check nodes and variable nodes is usually very large, so the consumption of the address generator is always very large. In this design, the address generator two parts, a ROM stores the initial address of every sub-matrix and an address-shifter which consisted of an adder and a mode 128 counter. The address shifter generates the address for each RAM by adding the initial address and the number of the counter together, so that the costing of the address generator can be lowered.

Moreover, to short the decoding time the decoder consumes and balance the resource costing of the decoder, we choose a two-parallel scheme. In this scheme, the variable nodes update model and the check nodes update model have two inputs with one computing core, so that the decoding time will be half of serial decoder in the same code length with little increase in resource costing.

Finally, to fit the special two-input update model, RAM_r and RAM_q will be a two-input and two-output RAMs, so that they can input or output two values at the same time. What is more, in order to let RAM_r outputs 0s at the first iteration, the reset port of this block RAM should be enabled. And after the first iteration, the reset port of it will be disabled again.

4. Results

A parallel low-costing LDPC decoder is designed and implemented on a Xilinx Virtex-4 FPGA platform. When the number of quantization bits and the max iterations are set to 6 and 25 respectively, the bit error rate (BER) performance of the decoder is shown as Figure 5 and Table 2.

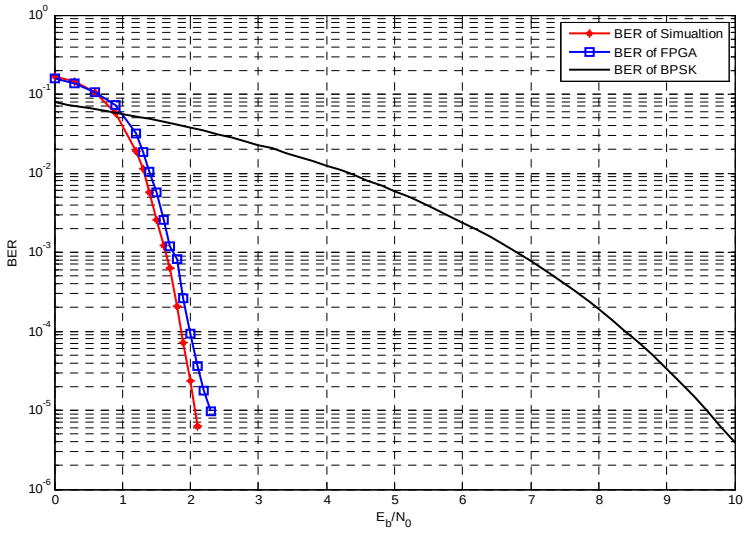

Figure 5. The BER Performance of the (2560,1024) LDPC Decoder

Table 2. The Coding Gain of (2560,1024) LDPC

BER	BPSK (Eb\N0\dB)	Coding Gain in Simulation (Eb\N0\dB)	Coding Gain in FPGA (Eb\N0\dB)
1e-2	4.32	3.01	2.90
1e-3	6.79	5.16	5.05
1e-4	8.40	6.53	6.41
1e-5	9.58	7.52	7.29

As we can see, the coding gain of the decoder we proposed in this paper is over 7.2 dB when BER is 1e-5, which is very close to the coding gain of the LDPC code we simulated on MATLAB. It means that, with the 6 quantization bits and 30 max iterations, this LDPC decoder can make good use of the performance of (2560, 1024) LDPC code.

The costing of the decoder proposed in this paper is shown in Table 3, and comparisons with some other decoders are also shown.

Table 3. The Costing of the Proposed Decoder

	Slices	4-inputs LUTs	16K RAMs/ROMs	Max Working Clock
Proposed	862 (3.5%)	1,084(2.2%)	10(3.1%)	181.242MHz
Reference[9]	7,755	22,014	8,555(Registers)	113MHz
Reference[10]	46,190	34,908	138	131.411MHz

Moreover, the throughput of the decoder can be calculated as below:

$$T_c = \frac{k \times T_D}{CLK} \tag{13}$$

Where T_C is the throughput of the decoder, k is the information bits in a codeword, T_D is the decoding clocks of the decoder, and CLK is the frequency of working clock. In the decoder we proposed, k=1024 bits, T_D= 208384clocks, CLK=180MHz, so the throughput of a single decoder T_C =8.5Mbps. But with the low-costing of the decoder, we can achieve a high throughput by using multi-parallel decoding strategy. For example, if we make full use of the FPGA we use, we can put 25 decoder on one FPGA, and the throughput can reach 212.5Mbps.

5. Conclusion

This paper proposes a LDPC decoding algorithm according to the features of FPGAs, and a low-costing LDPC decoder is implemented by using the GDL algorithm. The results shows that the LDPC decoder needs fewer resources compared again with other LDPC decoders, and the decoder can reach a high throughput by using multi-parallel decoding strategy. So the low-costing decoder is flexible for different applications.

References

[1] RG Galleger. Low density parity checkcodes. *IRE Trans. Info.Theory.* 1962; IT-8: 21-28.
[2] DJC MacKay, RM Neal. Near Shannon limit performance of low-density parity check codes. *Electron Lett.* 1996; 32(18): 1645-1646.
[3] Arnone LJ, Castineira Moreira J, Farrell PG. Field programmable gate arrays implementations oflow complexity soft-input soft-output low-densityparity-check decoders. *IET Communications*. 2012; 6(12): 1670-1675.
[4] VA Chandrasetty, SM Aziz. *A Multi-LevelHierarchical Quasi-Cyclic MatrixFor Implementation of Flexible Partially-Parallel LDPC Decoders*. IEEE International Conference on Multimedia and Expo (ICME), Barcelona. 2011: 1-7.
[5] Lei Yang, Hui Liu, CJ Richard Shi. Code Construction and FPGA Implementation of a Low-Error-Floor Multi-Rate Low-Density Parity-Check Code Decoder. *IEEE Transactions on Circuit and Systems*. 2006 53(4): 892-904.
[6] SM Aji, RJ McEliece. The Generalized Distributive Law. *IEEE Transactions on Information Theory*, 2000; 46(2): 325-343.
[7] FR Kschischang, BJ Frey, HA Loeliger. Factor Graphs and the Sum-Product Algorithm. IEEE Transactions on Information Theory. 2001; 47(2): 498-519.
[8] Mackay. Information Theory, Inference, and Learning Algorithm. 2006: 416 449.
[9] VA Chandrasetty, SM Aziz. *FPGA Implementation of High Performance LDPC Decoder using Modified 2-bit Min-Sum Algorithm.* Second International Conference on Computer Research and Development. 2010: 881-885.
[10] Yue Sun, Yuyang Zhang, Jianhao Hu, Zhongpei Zhang. *FPGA Implementation of Nonbinary Quasi-Cyclic LDPC Decoder Based On EMS Algorithm*. International Conference on Communications, Circuits and Systems, 2009: 1061-1065.