FPGA-Based High Throughput TDMP LDPC Decoder

Ruochen Liao(✉), Yuzhuo Fu, and Ting Liu

School of Microelectronics, Shanghai Jiao Tong University, Shanghai 200240, China
liaorc@sjtu.edu.cn

Abstract. In this paper, a high-throughput decoder architecture for quasi-cyclic low density parity check (QC-LDPC) codes is presented. Using the Normalized Min-Sum algorithm and the turbo-decoding message-passing algorithm, the proposed design expanded degree of parallelism to improve the throughput at a cost of hardware resource usage. Based on the proposed architecture, we implemented a (8176, 7154) Euclidian geometry-based QC-LDPC code decoder on a Xilinx Kintex7 (XC7K325T-2) board. The FPGA implementation results show that the decoder can achieve a total decoding throughput of 1.6 Gbps at the clock frequency of 105Mth at 10 iterations.

Keywords: Low-density parity-check · Turbo-decoding message-passing · Normalized min-sum · FPGA

1 Introduction

Low-Density Parity-Check (LDPC) codes were discovered by Gallager in 1962 [1]. Later in late 1990s, LDPC codes were rediscovered by MacKay and shown to approach Shannon capacity [2]. With excellent error control capacities, LDPC codes have been considered for emerging wireless communication standards, such as IEEE802.16e, IEEE802.11n, and DVB-S2 etc.

The original approach for LDPC decoders is known as Gallager's two-phase message-passing (TPMP) algorithm. Since TPMP algorithm need to decode iteratively using sum-product algorithm [1] at a high cost of calculation complexity, variations such as the min-sum algorithm and normalized min-sum algorithm [3] are also widely used. Previous works focused on the hardware implementations of TPMP algorithm with all kinds of architectures (e.g. fully parallel, partially parallel, and serial) have been proposed. However, the convergence speed of these methods are slow, which is troublesome in high-throughput applications. To address this problem, turbo-decoding message-passing (TDMP) algorithm [4] has been proposed. A single iteration in TPMP algorithm is divided into sub-iterations to accelerate convergence speed. The throughput is increased since less number of iterations is needed at the same bit error rate (BER).

The proposed architecture aims to expand degree of parallelism of a TDMP LDPC decoder. We take advantage of pipelining and parallel processing in the proposed high-throughput FPGA decoder. In additional with configurable depth of the block RAM in FPGA design, which allows the high bandwidth of message passing. Furthermore, the design carefully deals with the choosing of degree of parallelism to avoid data conflict

W. Xu et al. (Eds.): NCCET 2016, CCIS 666, pp. 94–101, 2016.
DOI: 10.1007/978-981-10-3159-5_9

when introducing pipelining. In result, the proposed decoder greatly improved parallel level in processing comparing with others. The improvement helps the decoder to achieve a high-throughput under a low clock frequency.

2 Previous Works

In this section, previous works with FPGA-based hardware designs are listed.

Chen Xiaoheng, Kang Jinyu and Lin Shu built a (8176, 7156) LDPC decoder with two specific optimizations called vectorization and folding [5]. With their efforts on taking advantage of configurable embedded memory in FPGA, the decoder achieved a total throughput of 713.8 Mbps at 15 iterations.

Wang Zhongfeng proposed a (8176, 7156) LDPC decoder based on Xilinx field programmable gate array Virtex-II 6000 [6]. The decoder employed an efficient nonuniform quantization scheme to reduce the size of memories storing soft messages. The results showed that Wang's work achieved a maximum decoding throughput of 172 Mbps at 15 iterations.

Xiang Bo in his work at 2011 implemented decoders for multiple LDPC code length ranging from 576 to 2304 [7]. Xiang's main effort was to include pipelining, block interleaving and nonzero sub-matrix reordering into his decoding technic. By Xiang's reordering feature, memory conflict was reduced when using pipeline in LDPC decoder. Xiang's best result achieved 955 Mbps at 10 iterations with a clock frequency of 214 MHz.

Previous works on LDPC decoder using TDMP algorithm share a common problem, which is that they all need a relatively high clock frequency to achieve a high throughput. For TDMP algorithm introduces potential memory conflict when processing multiple rows parallely, the dilemma between frequency and throughput appears. Our work aims at analysing the best degree of parallelism in TDMP algorithm. Therefore the proposed decoder reaches a high throughput at a relative low clock frequency.

3 High-Throughput Parallel LDPC Decoder Architecture

3.1 Decoding Algorithm

The proposed decoder uses the same algorithm as the one in work [8], which is the min-sum turbo decoding message passing algorithm. With TDMP algorithm, the convergence speed is faster than its opponent TPMP algorithm. By using min-sum algorithm, the calculation complexity of updating log-likelihood ratios is reduced.

3.2 Decoder Overview

In this section, the overview architecture of the proposed decoder will be shown. And the function of each part will be introduced.

Generally, the proposed LDPC decoder consist of four main parts. Four parts and the interconnects are shown in Fig. 1.

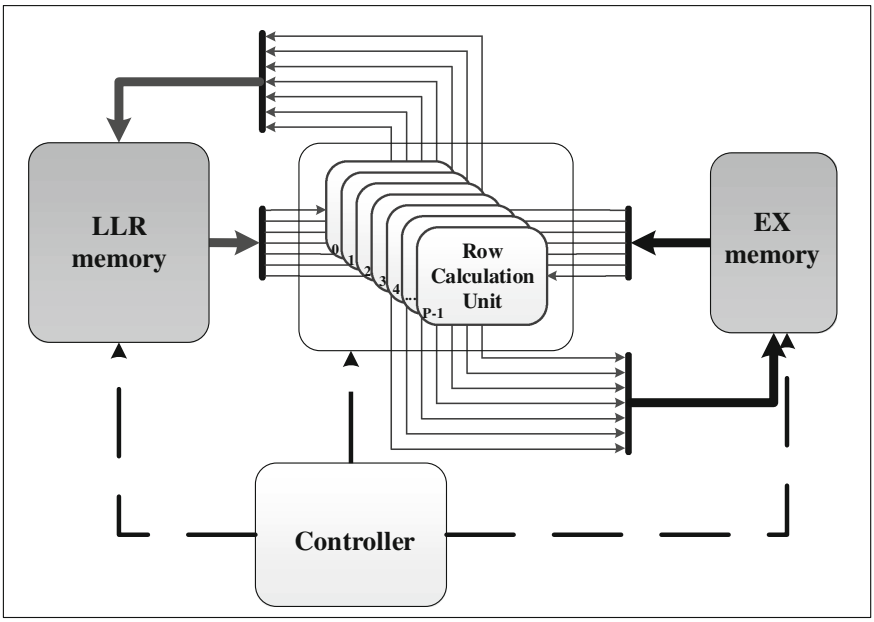

Fig. 1. LDPC decoder overview

The function of each part:

- LLR memory: The log-likelihood ratio memory. This module stores log-likelihood ratio of all the check nodes. This module also deals with the input and output data flow.
- EX memory: The extrinsic information memory. This module stores the extrinsic information passing from check nodes to variable node.
- Controller: The control module of decoder. This module generates read and write addresses for all the memories.
- Row calculation Unit: The row information update module. This is the main calculation module of the decoder, which does all the computation process and update the log-likelihood ratio and extrinsic information of each row in every iteration. The proposed parallel and pipeline structure is implemented in this part. Figure 2 shows that we implemented a number of P row calculation units in total, which means P rows of check nodes can be calculating in parallel.

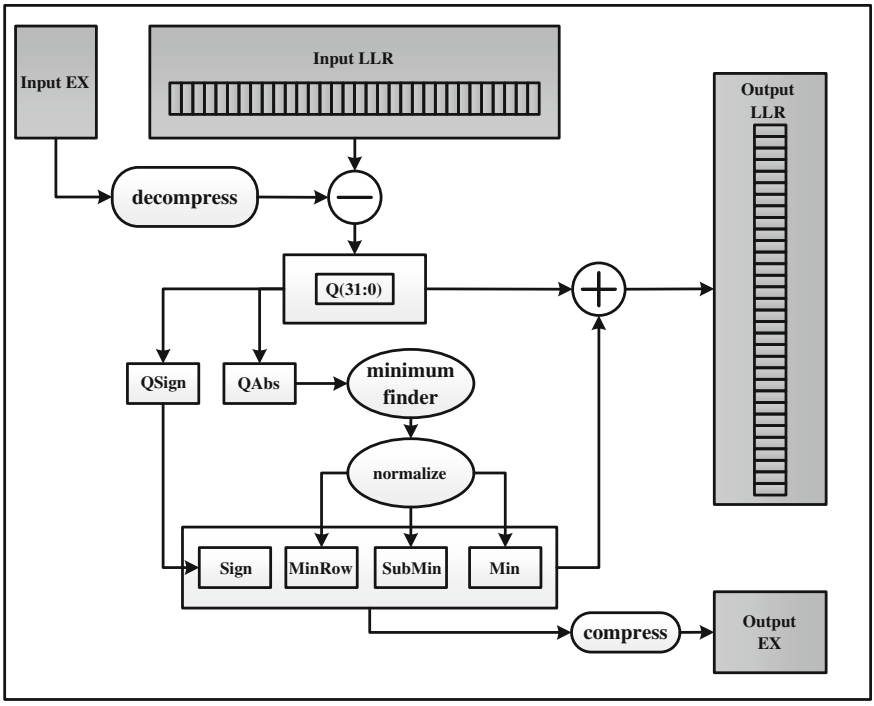

Fig. 2. Row calculation unit

3.3 Parallel Decoder Core Architecture

This is the structure for a single row calculation unit in Sect. 3.2.

Basically the update process is divided into three stages. The first stage deals with the data reading and decompression of extrinsic information. The second stage handles the main calculation process of check node unit (CNU). The last stage compresses the data and stores the updated data back to memory.

3.4 Parallel Decoder Core Architecture

During updating one log-likelihood ratio, three steps are needed by LDPC decoder: read original data from memory, calculate new log-likelihood ratios and extrinsic information, write back the updated data to memory. In Fig. 3, read or write cycle is assumed to be T_c (T_c also stands for time of a clock cycle), and the delay of calculation module is assumed to be X ns. Thus the cycle delay of updating one data is,

$$T_{path} = \left\lceil \frac{2T_c + X}{T_c} \right\rceil \tag{1}$$

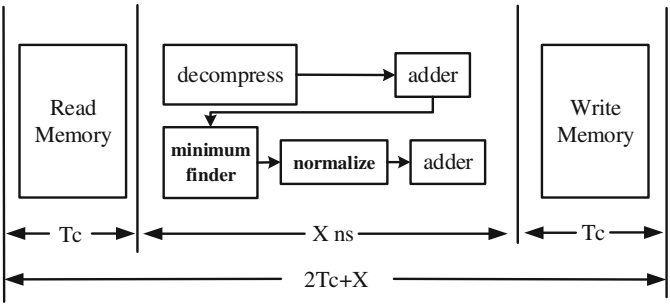

Fig. 3. Delay path for updating one log-likelihood ratio

In the proposed design, pipelining and parallel processing are introduced to accelerate the decoding speed. With parallel processing feature, P is set for the parameter of the number of sub-matrix processing at the same time. And when introducing pipelining into the design, memory conflict appears to be a new problem.

In the design, memory conflict is prevented during one iteration, but it is hard to avoid when algorithm runs form one sub-matrix to the next one. Figure 4 tells that when memory conflict occurs our pipeline should stall for cycles to ensure the algorithm runs properly and the stall cycles is set to T_{stall}. Then the total cycles of updating one iteration T_{total} is,

$$T_{total} = 2N_{iter}\left(T_{stall} + \left\lceil \frac{511}{P} \right\rceil\right) + T_{path} - 1 \tag{2}$$

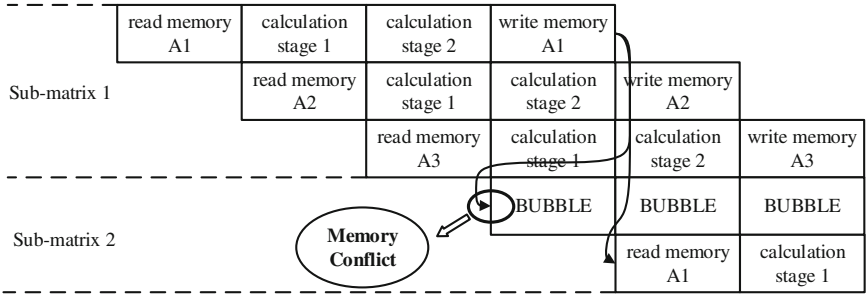

Fig. 4. Potential memory conflict in pipelining

Since we want to avoid memory conflict during one iteration, the offset distance in one sub-matrix should be taken into consideration. We define $offset_1$ and $offset_2$ to be the distance between 1's in one sub-matrix and define a parameter L,

$$L = \min(offset_1 - offset_2,\ 511 - (offset_1 - offset_2)) \tag{3}$$

For (8176, 7154) LDPC code, $L \geq 104$. Thus to avoid memory conflict, we need

$$T_{path}P \leq L \tag{4}$$

On the other hand, $Throughput = \frac{f_{clk}len}{T_{total}}$. And apparently, $len = 7136$ in (8176, 7154) code.

The design needs a total throughput no less than 1.6 Gbps. Therefore, with the analysis above, we get one solution that $P = 26$, $f_{clk} \approx 100$ MHz, $T_{path} = 4$, $T_{stall} = 3$.

3.5 Quantization

Since quantization of log-likelihood ratios affects the memory usage of LDPC decoder and the path delay of data processing, it is reasonable to choose a shorter quantization mode for our design (Table 1).

Table 1. Coding gain analysis

Coding gain (dB)		Iteration						Resource util. (Slice)
		5	6	7	8	9	10	
Quantization	Float	7.17	7.24	7.24	7.24	7.30	7.32	/
	7bit [2, 4]	7.12	7.20	7.17	7.22	7.24	7.25	45346
	6bit [2, 3]	7.00	7.08	7.05	7.10	7.11	7.13	39741
	5bit [2, 2]	6.69	6.74	6.78	6.78	6.78	6.79	33118

When processing with 10 iterations, 5bit quantization results in a 6.79 dB coding gain, only 0.4 dB worse than a float number decoder. But with 5bit quantization, the slice resource utilization will drop 26.9%. Since a 6.79 dB coding gain is enough for our case, we choose a 5bit quantization.

4 Implementation and Performance Evaluation

4.1 Implementation Platform

The decoder is implemented on a Xilinx Kintex7 (XC7K325T-2FFG900) board under development environment Xilinx Vivado. The design can meet the timing constraint to run under 105 MHz. To test the function of our decoder, we used a host computer and the PCI-E interface to control the decoder and transmit data with it.

Considering the data transmission speed would not match the design throughput of our LDPC decoder, we add additional input and output FIFO to our decoder to test its highest performance (Fig. 5).

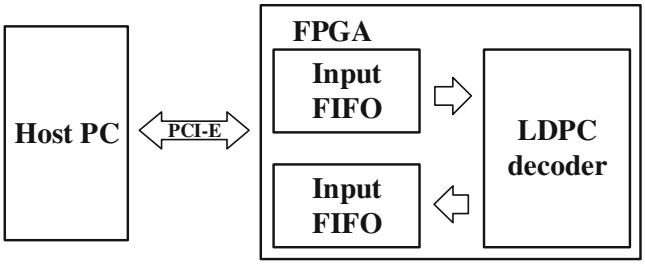

Fig. 5. System overview for decoder testing

4.2 Implementation Result and Resource Utilization

We prototyped the proposed high-throughput LDPC decoder architectures on Xilinx Kintex7 (XC7K325T-2FFG900) device. The prototype is verified under a 1e10 bits LDPC data decoding test. We also tested our design in variety working frequency. All the FPGA resource utilization result after implementation is given in Table 2.

Table 2. FPGA resource utilization

Clock frequency	75 MHz	85 MHz	95 MHz	105 MHz
FF	165067	165067	175649	175650
LUT	132214	132222	151501	152761
BRAM	221	221	221	221

4.3 Comparison with Other Works

Table 3 shows the throughput result of proposed design and some of previous works. Comparing with other woks, the proposed design greatly improves degree of parallelism and achieves a total throughput of 1.6 Gbps only requiring a clock frequency of 105 MHz. To have a more reasonable comparison, we introduce this so called Normalized Throughput, which is defined as,

Table 3. Comparision

Parameter	Wang [6]	Xiang [7]	Chen [5]	Proposed
FPGA	Virtex-II 6000	CMOS	XC4VLX-160	XC7K325T
Clock frequency (MHz)	193	214	212.2	105
Degree of parallelism	2	2	4	26
Iteration	15	10	15	10
Throughput (Mbps)	199.2	728	713.8	1628.8
Normalized Throughput (Mb)	15.5	34.0	50.5	155.1
FPGA utilization (slice)	23052	-	17857	38213

$$T_{pN} = T_p \times N_{iter}/f_{clk} \tag{5}$$

Comparison between Normalized Throughput takes iteration number and frequency into consideration. According to the result table, the proposed decoder has 3 times of Normalized Throughput comparing with others.

5 Conclusion

We present a high-throughput (8176, 7154) LDPC decoder with expansion of degree of parallel. Our work takes the advantage of pipelining and parallel processing to implement TDMP algorithm. When compared with previous works on LDPC decoder, our result achieves a two-times improvement in normalized throughput, which indicate the proposed architecture have a high throughput in low clock frequency system. Another main contribution of this work is to give an analysis on the selecting of best degree of parallelism for the (8176, 7154) LDPC code. The same method can also be applied to other LDPC codes to design large parallel high-throughput decoders.

Acknowledgements. This work is supported by the National Science Foundation of China (Grant No. 61373032, Grant No. 61472244) and Innovation Program of Shanghai Municipal Education Commission (Grant No. 14ZZ018).

References

1. Gallager, R.G.: Low-density parity-check codes. IRE Trans. Inf. Theory **8**(1), 21–28 (1962)
2. MacKay, D.J.C.: Good error-correcting codes based on very sparse matrices. IEEE Trans. Inf. Theory **45**(2), 399–431 (1999)
3. Fossorier, M.P.C., Mihaljević, M., Imai, H.: Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. IEEE Trans. Commun. **47**(5), 673–680 (1999)
4. Mansour, M.M.: A turbo-decoding message-passing algorithm for sparse parity-check matrix codes. IEEE Trans. Sign. Process. **54**(11), 4376–4392 (2006)
5. Chen, X., et al.: Memory system optimization for FPGA-based implementation of quasi-cyclic LDPC codes decoders. IEEE Trans. Circuits Syst. I Regul. Pap. **58**(1), 98–111 (2011)
6. Wang, Z., Cui, Z.: Low-complexity high-speed decoder design for quasi-cyclic LDPC codes. IEEE Trans. Very Large Scale Integr. Syst. **15**(1), 104–114 (2007)
7. Xiang, B., et al.: An 847–955 Mb/s 342–397 mW dual-path fully-overlapped QC-LDPC decoder for WiMAX system in 0.13 m CMOS. IEEE J. Solid-State Circuits **46**(6), 1416–1432 (2011)