Optimization of FPGA-based LDPC decoder using high-level synthesis

Geon Choi
Hanyang University
82-2-2220-4701
ggrice@hanmail.net

Kyeong-Bin Park
Hanyang University
222, Wangsimni-ro, Seongdong-gu
Seoul, Republic of Korea
82-2-2220-4701
lay1523@naver.com

Ki-Seok Chung
Hanyang University
82-2-2220-4701
kchung@hanyang.ac.kr

ABSTRACT

Low Density Parity Check (LDPC) codes are widely used in various communication and storage systems due to outstanding error correcting capability. In this paper, we present a Field Programmable Gate Array (FPGA) implementation of the LDPC decoder using High-Level Synthesis (HLS). Because HLS can synthesize a hardware implementation from a high-level description, it is very effective in reducing design time, and in exploring various design alternatives. One of the biggest advantages of FPGAs is flexibility, and therefore, HLS for FPGAs is widely adopted as a good hardware synthesis method. In this paper, we describe an LDPC decoder in high level language, and a HLS tool called SDSoC is used to synthesize the decoder. The proposed design is a serial LDPC decoder that requires smaller amount on hardware resource and power consumption than the conventional design. The major drawback of a serial decoder is slow speed. To overcome such drawback, optimization techniques such as array partitioning, loop unrolling, pipelining methods and fixed-point conversion are applied. With the application of these techniques, the decoding speed of the proposed implementation is 8.11 times and 2.79 times faster than that of a non-optimized implementation and that of a software-based LDPC decoder, respectively.

CCS Concepts

• **Hardware → High-level and register-transfer level synthesis;**
• **Computer systems organization → System on a chip;**

Keywords

Error Correcting Code; Field Programmable Gate Array; High Level Synthesis; Low Density Parity Check; SDSoC;

1. INTRODUCTION

The Low-Density Parity Check (LDPC) code is one of forward error correction block codes, and it corrects errors by carrying out decoding operations iteratively [1]. LDPC has been used communication standards such as IEEE 802.11 (Wi-Fi) and DVB-S2 [2] [3]. Also, it has been employed in flash storage systems [4] [5]. The structure of LDPC is defined by a matrix called Parity Check Matrix (PCM). PCM is an ultra-sparse matrix meaning that the number of zero elements is a lot more than the number of non-zero elements. LDPC codes are typically decoded by a message-passing algorithm, which iteratively exchanges messages, and the performance of the LDPC code is known to be very close to the Shannon limit [6].

In this paper, we propose an implementation of a serial LDPC decoder on a System-On-Chip (SoC) platform that consists of Central Processing Unit (CPU) and Field Programmable Gate Array (FPGA). With rapid advances in density and performance, FPGAs now replace Application Specific Integrated Circuit (ASIC) in some SoC applications. Today, there are quite a few SoC platforms that consist of multi-core CPU and high-end FPGA.

There are many studies for designing LDPC decoders on FPGAs [7]. Most designs targeting FPGAs are synthesized from Register Transfer Level (RTL) codes. Describing a design at RTL takes long time and much effort. To reduce design time and effort, Xilinx, the leading FPGA manufacturer, announced a High-Level Synthesis (HLS) tool called SDSoC. SDSoC makes it possible for developers to implement a design on an FPGA from high level description such as C/C++. However, it is not straightforward to synthesize a design of good quality from high level descriptions. Therefore, it is very important to apply appropriate optimization techniques. In this paper, we will address optimization techniques that are applied to our proposed LDPC decoder to achieve high performance.

The remainder of this paper is organized as follows. First, we briefly address LDPC and its decoding process in Section 2. In Section 3, we explain applied optimization methods and the architecture of the proposed LDPC decoder. Experimental results and analysis will be given in Section 4. Section 5 will conclude this paper.

ICCIP 2018, November 2–4, 2018, Qingdao, China
© 2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6534-5/18/11…$15.00
http://doi.org/10.1145/3290420.3290441

2. LOW DENSITY PARITY CHECK CODES

2.1 Organization of LDPC

The structure of LDPC is defined by PCM. The number of rows in PCM corresponds to the number of parity bits (M), and the number of columns in PCM corresponds to the number of transmitted bits (N). K denotes the number of data bits, which implies that $M = N - K$. The code rate R can be defined as $R = 1 - M/N$ and correspondingly, $0 < R < 1$. The number of non-zero elements in a row and that in a column in PCM are called as row degree and column degree, respectively. The row degree denotes the number of variable nodes connected to a check node and therefore, it corresponds to the check node degree. The column degree denotes that the number of check nodes connected to a variable node and thus, it corresponds to the variable node degree. If all the variable node degrees are the same for each column and if all the check node degrees are the same for each row, corresponding LDPC code is called as regular LDPC.

Min-sum algorithm

$L_{init,n}$: log likelihood ratio on nth node

$C_{m,n}$: check node message from mth check node to nth variable node

$V_{m,n}$: variable node message from mth variable node to nth check node

m = {1, …, M}, n = {1, …, N}

α : sign of node message

β : absolute value of node message

• Check node processing

$$C_{m,n} = \left(\prod_{n' \in \mathrm{H}(m) \backslash \{n\}} \alpha_{n',m} \right) \bullet \min_{n' \in \mathrm{H}(m) \backslash \{n\}} \beta_{n',m}$$

$$\alpha_{n',m} = sign(V_{n,m}),\ \beta_{n',m} = |V_{n,m}|$$

• Variable node processing

$$V_{n,m} = L_{init,n} + \sum_{m' \in \mathrm{H}(n) \backslash \{m\}} C_{m' \to n}$$

• Hard decision and syndrome checking

$$\hat{c} = \left(\left(L_{init,n} + \sum_{m \in \mathrm{H}(n)} C_{m \to n} \right) \geq 0 \right) \ ?\ 1 : 0$$

$$\hat{c} \cdot \mathrm{H}^T = 0$$

Figure 1. Computation steps in Min-sum algorithm

2.2 Decoding with LDPC Codes

LDPC codes are typically decoded by a message-passing algorithm, which iteratively exchanges messages. There are many different message-passing algorithms. Among them, the Sum-Product Algorithm (SPA) is known to have the most powerful decoding capability; however, high decoder complexity is a serious concern. Thus, hardware implementations are based on another method called Min-Sum Algorithm (MSA) because a good trade-off between satisfactory error correction performance and relatively low design complexity can be achieved. So, the proposed LDPC decoder of this paper employs MSA. Figure 1 shows three key operations in MSA: check-node processing, variable-node processing, and hard decision with syndrome checking.

3. IMPLEMENATATION OF LDPC DECODER

3.1 Optimization Methods

LDPC decoding requires a lot of iterative operations and memory accesses, so it is very crucial to accelerate loop operations while minimizing performance degradation due to heavy memory access. In this paper, we apply array partitioning, loop unrolling and pipelining.

3.1.1 Array Partitioning

Array partitioning means that a matrix is divided into small sub-matrices and store each sub-matrix into a lane of the internal FPGA memory block called Block Random Access Memory (BRAM). As shown in Figure 2, the array partitioning enables parallel memory accesses to increase memory access throughput. In the proposed serial LDPC decoder, each variable node operation is executed by one Variable Node Processor (VNP), and each check node operation is executed by one Check Node Processor (CNP). The array partitioning is not effective to improve performance when the node degree is small, so the array partitioning is only applied to CNP in the proposed decoder because the variable node degree is small.

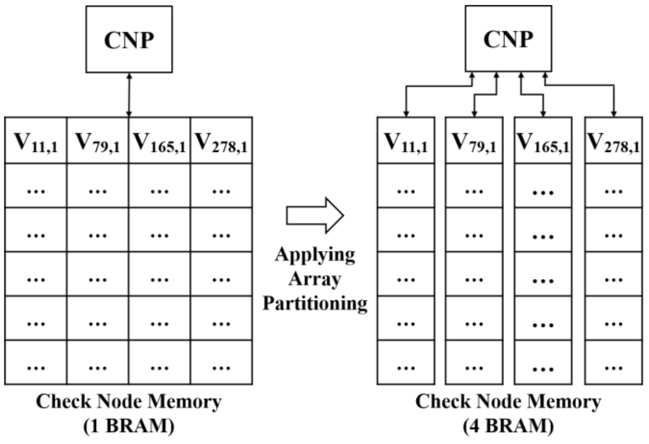
CNP
V11,1
V79,1
V165,1
V278,1
Applying Array Partitioning
CNP
Check Node Memory (1 BRAM)
Check Node Memory (4 BRAM)

Figure 2. Check node memory architecture in applying array partitioning

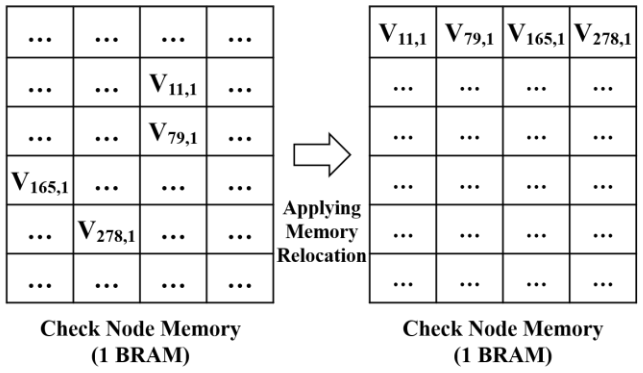
V11,1
V79,1
V165,1
V278,1
Applying Memory Relocation
Check Node Memory (1 BRAM)
Check Node Memory (1 BRAM)

Figure 3. Applying memory relocation in check node memory

```
short v_sign[C_NODES][C_DEGREE];
#pragma HLS array partition variable=v_sign block factor=6 dim=2

CHECK_SIGN_VAL_DIV:
for (short k = 0; k < C_NODES; k++) {
        for (short i = 0; i < C_DEGREE; i++) {
                v_sign[k][i] = (v_message[c_address[6*k+i]]<0);
        }
}
```

Figure 4. Partial code of memory relocation and array partitioning

To apply array partitioning to the check node memory, data relocation is needed. Because check nodes are irregularly connected to variable nodes, the data should be relocated in the order of memory accesses in the check node operation. Memory relocation is a 2-dimensional array mapping that stores the data in the order of variable nodes that are connected to the check node. Figure 3 shows an example data store in the check node memory. "$V_{11,1}$" denotes a message from the 11th variable node to the 1st check node. It is assumed that the 11th, 79th, 165th and 278th variable nodes are connected to the 1st check node in a Tanner graph to represent a PCM. Without data relocation, messages that a check node needs are stored in the order that variable nodes generate them. Then, applying array partitioning may not be effective. Therefore, by applying memory relocation, the values that are necessary in the *n*th check node operation are placed in the accessed order on the *n*th row of the check node memory. The *n*th row of check node memory is supposed to be accessed in the *n*th check node operation and the column position of the check node memory corresponds to the order of the variable node that is connected to the check node. Without array partitioning, the check node memory is accessed as many times as the check node degree for each check node operation. Figure 4 shows partial code about applying memory relocation and array partitioning. And figure 2 shows the check node memory architecture when memory relocation and array partitioning are applied. After applying the two methods, BRAMs are built as many as the check node degree, and CNP can access to the multiple BRAMs at the same time. Correspondingly, the memory throughput is improved.

3.1.2 Loop Unrolling

Loop unrolling is a method of reducing the number of loop iterations by unrolling loop bodies of multiple iterations into a body of a single iteration. Through loop unrolling, loop control overhead is reduced. Each unrolled loop body may be processed in parallel, and the synthesis tool automatically generates the hardware control logic circuit. Loop unrolling is applied in a part of VNP and the initialization phase. Then, the subtraction operation of VNP and the memory access operation of the initialization are performed in parallel.

3.1.3 Pipelining

Pipelining splits a process into multiple stages and executes the stages in an overlapping fashion. if each stage is independent, speed-up of as much as the number of stages can be achieved even without having to add multiple sets of processing units. Thus, pipelining is essential for serial LDPC decoders because there is only one CNP and one VNP. Therefore, in the proposed LDPC architecture, the initialization process, key processes of CNP and VNP, and syndrome checking are carried out in a pipelined manner. When the pipelining is applied, multiple memory accesses can occur at the same time. We resolve this issue by applying the array partitioning and memory relocation methods.

3.1.4 Fixed-Point Conversion

In general, exchanged messages in LDPC decoding are represented by floating point numbers. Obviously, handling floating point numbers are complicated and computationally expensive. Therefore, floating point numbers are replaced by fixed-point numbers in the proposed LDPC decoder. This fixed-point may cause degradation of BER performance. So, a fixed-point scaling factor of 32 is applied to minimize degradation and fixed-point conversion is applied by multiplying floating point numbers by 32.

3.2 Architecture of LDPC on an SoC Platform

The proposed LDPC decoder is implemented on an SoC platform called Xilinx Zynq [8]. Zynq consists of Processing System (PS) which is an ARM dual-core cortex A9 CPU and Programmable Logic (PL) which is Kintex-7 FPGA. PL consists of BRAM, configurable logic blocks and Digital Signal Processor (DSP). Configurable logic blocks are composed of Flip-Flops (FF) and Look-Up Tables (LUTs). Figure 5 shows how functional blocks of the proposed LDPC decoder are mapped to PS and PL on Zynq. PS reads PCM, and it generates message addresses and then, stores them in Dynamic Random Access Memory (DRAM). Message addresses and message data are sent to BRAM in PL, and PL carries out iterative operation processing. After the decoding operation, decoded bits are sent to DRAM in PS. The processes from the initialization to the syndrome check and the control logic between PS and PL are automatically synthesized by SDSoC.

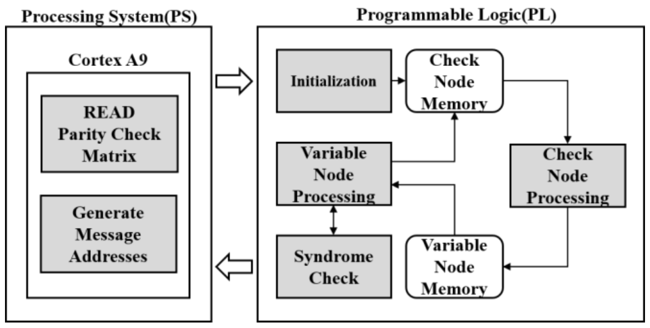

Figure 5. The proposed LDPC decoder architecture

4. RESULTS AND ANALYSIS

The performance of the proposed serial LDPC decoder is evaluated with regular LDPC codes for the IEEE 802.11 Wi-Fi standard. The PCM has the code length of 816, the data of 408 bits and the code rate of 0.5. The operating clock frequency of PS and that of FPGA are 1GHz and 100MHz, respectively.

4.1 Decoding Time Comparison

We compare the proposed design (PL_{OPT}) with a non-optimized decoder (PL_{NON}) and a software decoder (PS_{SW}). PL_{NON} is a hardware implementation with only the fixed-point conversion applied while PL_{OPT} is the proposed hardware implementation with all of the aforementioned optimization methods applied. Both decoders are described in C language. PS_{SW} is a software decoder where the entire execution is conducted by PS. Table 1. summarizes the comparison results. When Signal to Noise Ratio (SNR) is low, the decoder needs more effort to correct errors. It is confirmed that the performance improvement increases as SNR gets lower, and it is because the decoding iteration count increases when SNR is low. The speed of PL_{OPT} is 8.11 times faster than PL_{NON} and 2.79 times faster than PS_{SW} when SNR is the lowest, 1. The experimental results confirm that the proposed design is best in terms of the decoding speed.

Table 1. Comparison of decoding time per 1 frame.

SNR	Time(ms)			Speed up	
	PL		PS_{SW}		
	PL_{OPT}	PL_{NON}		vs PL_{NON}	vs PS_{SW}
1	3.687	29.919	10.296	8.11	2.79
1.5	2.472	18.729	6.344	7.58	2.57
2	1.448	8.904	3.015	6.15	2.08
2.5	1.102	5.411	1.685	4.91	1.53
3	0.968	4.128	1.219	4.27	1.26
3.5	0.920	3.377	0.958	3.67	1.04

4.2 Usage of Hardware Resource

Table 2. shows the comparison of the amount of hardware resource. Fluctuation rate is the hardware resource variation rate of PL_{OPT} based on PL_{NON}. In PL_{OPT}, the usage of BRAM, LUT and FF are all increased compared to PL_{NON}. The array partitioning method causes the amount of necessary BRAM to increase, but the size of data is decreased due to the fixed-point conversion. The usage of LUT increases, because SDSoC synthesizes the decoder with the way that more LUTs are utilized when applying array partitioning and memory relocation.

Table 2. Hardware resource.

Hardware resource	PL_{OPT}	PL_{NON}	Difference	Fluctuation rate
BRAM	28.5	27.5	1	3.64%
LUT	9,208	8,459	749	8.85%
FF	13,201	13,197	4	0.03%

4.3 BER Performance of LDPC decoders

Figure 6 shows Bit-Error Rate (BER) performance with respect to SNR. The maximum iteration count is set to 50 and the total number of frames is set to 100,000 on each SNR. To find an appropriate scaling factor for fixed-point type conversion, we have tried several different scaling factors. The BER performance degradations are 17.1%, 5.5%, 2.4%, and 1.4% when the scaling factors are 4, 8, 16, and 32, respectively. To minimize the BER performance degradation, the scaling factor is set to 32. Figure 6 shows the BER performance of the proposed decoder compared with that of the software decoding with floating-point numbers, and two BER curves are almost identical. So, it is confirmed that there is almost no BER performance degradation due to the fixed-point type conversion.

5. CONCLUSION

In this paper, we propose an LDPC decoder that is implemented on an SoC platform. Computationally intensive part of the decoder is synthesized by a high-level synthesis tool called SDSoC, and mapped on an FPGA device in the SoC platform. Optimization techniques such as array partitioning, loop unrolling, pipelining methods and fixed-point conversion are applied. Experimental results show that performance improvement is better when SNR is lower. When compared to a non-optimized hardware decoder and a software decoder, the speed-up's of the proposed design are up to 8.11 and 2.79, respectively.

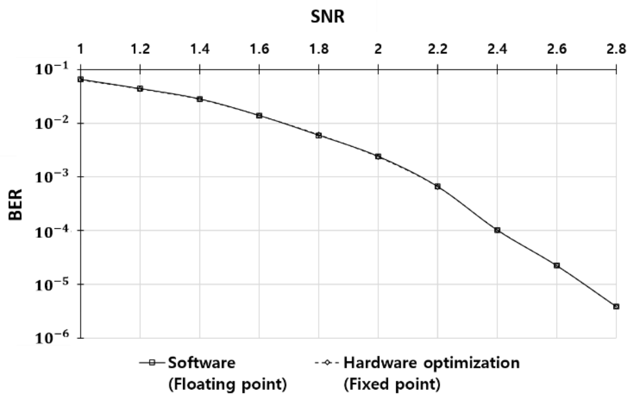

Figure 6. BER performance comparison between fixed-point and floating-point operation

6. ACKNOWLEDGMENTS

This research was supported by the KIAT(Korea Institute for Advancement of Technology) grant funded by the Korea Government(MOTIE : Ministry of Trade Industry and Energy). (No. N0001883, HRD Program for Intelligent semiconductor Industry)

7. REFERENCES

[1] Gallager, R. G., 1962. Low-density parity-check codes. *IRE Trans*. Inf. Theory, vol. IT-8, no. 1, pp. 21-28.

[2] *ETSI*, 2004. Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications, EN 302 307, V1. 1. 1.

[3] IEEE P802.11n/TM-2009, 2009. *IEEE standard for information technology part 11*: wireless LAN medium access control (MAC) and physical layer (PHY) specifications.

[4] Kong L., Wen J., Han G., Zhao S., Jiang M., and Zhao C., 2016. Quantization and reliability-aware iterative majority-logic decoding algorithm for LDPC code in TLC NAND flash memory, in Proc. of 2016 8th *International Conference on Wireless Communications & Signal Processing (WCSP)*, pp. 1–5.

[5] Ho K.-C., Chen C.-L., and Chang H.-C., 2016. A 520k (18900, 17010) array dispersion LDPC decoder architectures for NAND flash memory, *IEEE Trans*. Very Large Scale Integr. Syst., vol. 24, no. 4, pp. 1293–1304.

[6] Chung S.-Y., Forney G. D., Jr., Richardson T. J., and Urbanke R., 2001. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit, *IEEE Commun*. Lett., vol. 5, no. 2, pp. 58_60.

[7] Hailes P., Xu L., Maunder R. G., Al-Hashimi B. M., and Hanzo L., 2016. A survey of FPGA-based LDPC decoders*, IEEE Commun*. Surveys Tuts., vol. 18, no. 2, pp. 1098_1122, 2nd Quart.

[8] https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html, 2015.