Algorithm and Hardware Co-design for Deep Learning-powered Channel Decoder: A Case Study

Boyang Zhang*
Rutgers University
bz207@scarletmail.rutgers.edu

Yang Sui*
Rutgers University
yang.sui@rutgers.edu

Lingyi Huang
Rutgers University
lh670@scarletmail.rutgers.edu

Siyu Liao†
Amazon
liaocs2008@gmail.com

Chunhua Deng†
ScaleFlux
chunhua.deng@scaleflux.com

Bo Yuan
Rutgers University
bo.yuan@soe.rutgers.edu

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643510

***Abstract*—Channel decoder is a key component module in many communication systems. Recently, neural networks-based channel decoders have been actively investigated because of the great potential of their data-driven decoding procedure. However, as the intersection among machine learning, information theory and hardware design, the efficient algorithm and hardware co-design of deep learning-powered channel decoder has not been well studied.**

This paper is a first step towards exploring the efficient DNN-enabled channel decoders, from a joint perspective of algorithm and hardware. We first revisit our recently proposed doubly residual neural decoder. By introducing the advanced architectural topology on the decoder design, the overall error-correcting performance can be significantly improved. Based on this algorithm, we further develop the corresponding systolic array-based hardware architecture for the DRN decoder. The corresponding FPGA implementation for our DRN decoder on short LDPC code is also developed.

***Index Terms*—channel decoder, belief propagation, deep neural network, hardware architecture**

I. Introduction

Because of their strong error-correcting capability, channel codes have been wildly used in practice to improve the reliability of data transmission. Nowadays various types of channel codes, e.g., low-density parity-check (LDPC) [1], polar [2], Turbo [3], BCH [4] and convolutional codes [5], have been popularly adopted in tremendous wired and wireless systems such as 3G/4G/5G, WIFI/Bluetooth, optical communication and deep space communication etc.

Recently, motivated by the unprecedented success of AI techniques (especially deep neural networks (DNNs)) in various science and engineering applications, data-driven channel coding has become a very attractive solution. To date, numerous DNN-enabled channel codecs, a.k.a., neural channel decoder, for different types of channel codes have been reported in the literatures [6]–[17]. Despite their difference in the technical details, these state-of-the-art works share the same key idea – unfolding the classical iterative belief propagation (BP) decoding procedure along the time dimension to form a feedforward neural network. Within this design framework, the important scaling parameters of BP algorithm, which were empirically determined before, can now be trained from the data via the standard backward propagation. Such data-driven parameter determination, evidently, brings better error-correcting performance for the neural channel decoders.

Despite the current research prosperity of neural channel decoders, several fundamental questions have not been fully answered yet. For instance, whether we can further improve the performance of the neural channel decoder beyond the currently obtained progress or not? Meanwhile, considering the channel decoders are typically implemented in the format of customized hardware (e.g.. FPGA or ASIC) to satisfy the real-time requirement of communication systems [18]–[23], how should we explore and develop the corresponding hardware architecture for the neural channel decoders in this new design paradigm?

This paper is a first step towards exploring the uncharted territory of efficient DNN-powered channel decoder design when both the task performance (e.g., bit-error rate (BER)) and the hardware performance (e.g., latency and power) should be co-considered. More specifically, built upon our recently proposed low-cost high-performance doubly residual neural (DRN) decoder [24], we investigate the efficient hardware architecture design for channel decoder from the perspective of AI hardware accelerator – a research field that has owned very mature design methodology and comprehensive optimization techniques. We believe such a perspective, which is not typically adopted in the channel decoder hardware (and the wider VLSI DSP) community, may unlock some new research insights and opportunities. Although our current exploration is only a case study for short LDPC decoder, it could be potentially used in other types of channel decoders as well, especially for block codes.

The rest of this paper is organized as follows. Section II gives a brief introduction of channel codec, BP and neural BP decoding algorithms. Our recently proposed DRN decoder is revisited and described in Section III. Section IV presents the example hardware architecture of DRN decoder for short LDPC codes as a case study. The experimental results are

* These authors contributed equally.
† This work was done when the author was with Rutgers University.

978-1-6654-4507-8/21/$31.00 ©2021 IEEE

reported in Section V. And Section VI draws the conclusions.

II. Preliminaries

A. Channel codes

In general, linear block code $\mathcal{C}$ can be used to transmit a k-bit binary message $\mathbf{m}$ over a noisy channel. To be specific, it performs an injective mapping on $\mathbf{m}$ to generate an n-bit codeword $\mathbf{x}$ via using a binary generator matrix $\mathbf{G}$ of size $k \times n$. Each $\mathbf{G}$ is associated with a binary parity-check matrix $\mathbf{H}$ with the size of $m \times n$, where $m = n - k$ and $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$.

Before transmission, the binary message $\mathbf{m}$ is encoded into the codeword $\mathbf{x} = \mathbf{m}\mathbf{G}$. During the transmission, $\mathbf{x}$ is corrupt by the channel noise and hence at the receiver end the received codeword changes to $\mathbf{r}$. To recover $\mathbf{x}$ from $\mathbf{r}$, different types of decoding strategies, such as hard-decision or soft-decision schemes, can be used for different specific channel codes.

Our case study focuses on LDPC codes, a type of channel codes that have been used in 5G standard. According to [25], a LDPC code can be defined via its Tanner graph $\mathcal{G}$. As illustrated in Fig. 1, a Tanner graph consists of a set of n variable nodes VNs V (that corresponds to the bits in $\mathbf{x}$) and a set of m check nodes CNs C (that corresponds to the parity-check equations). For simplicity, we use v to denote the v-th variable node in V and c to denote the c-th check node in C, respectively. VNs and CNs are connected via a set of edges $E = \{e_{(c,v)} = (c, v) : H(c, v) = 1, v \in V, c \in C\}$. Each $e_{(c,v)}$ connecting the c-th check node and the v-th variable node corresponds to one elements "$H(c, v) = 1$" in a given $\mathbf{H}$ matrix. Hence as indicated in Fig. 1, for each LDPC code its Tanner graph and $\mathbf{H}$ matrix is a one-to-one mapping. In addition, practical systems typically adopt regular LDPC codes, where the $\mathbf{H}$ matrix has the fixed number of "1"s (d_c) in each row and the fixed number of "1"s (d_v) in each column.

B. BP algorithm

Modern LDPC codes are typically decoded via belief propagation (BP) algorithm. As revealed by its name, the BP algorithm propagates the belief messages through the edges in $\mathcal{G}$ between VNs and CNs in an iterative way (see Fig. 1). To be specific, after the BP decoder receives the log-likelihood ratios (LLRs) $\boldsymbol{l} \in \mathbb{R}^n$ of the received message $\mathbf{r}$ as

$$l_v = \log \frac{Pr(x_v = 1|r_v)}{Pr(x_v = 0|r_v)}, \tag{1}$$

the VNs and CNs iteratively update the to-be-sent LLR messages during the entire BP process. Assume $u^t_{v\to c}$ and $u^t_{c\to v}$ denote the message transmitted from the v-th variable node to the c-th check node and from the v-th variable node to the c-th check node at the t-th iteration, respectively. Then the update principle of LLR message is as follows:

$$u^t_{v\to c} = l_v + \sum_{c' \in N(v)\backslash c} u^{t-1}_{c'\to v}, \tag{2}$$

$$u^t_{c\to v} = 2\tanh^{-1}\Big[\prod_{v' \in M(c)\backslash v} \tanh\big(\frac{u^t_{v'\to c}}{2}\big)\Big], \tag{3}$$

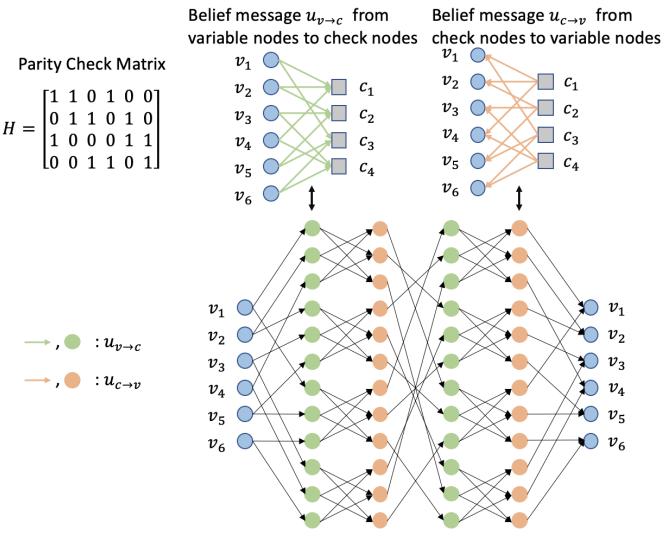

Fig. 1. Parity check matrix, corresponding Tanner graph and Trellis graph for an LDPC code.

where $N(v) = \{c \in C : e_{(c,v)} \in E\}$ and $M(c) = \{v \in V : e_{(c,v)} \in E\}$. Then, the final soft output after the t-th iteration can be calculated as:

$$s^t_v = l_v + \sum_{c' \in N(v)} u^t_{c'\to v}. \tag{4}$$

C. Neural BP algorithm

The classical BP algorithm described above can precisely calculate the posterior LLRs to provide the optimal decision of $\mathbf{r}$ if no girth exists in the Tanner graph. However, the practically used codes (such as LDPC) are always associated with the girth. In such a case, the classical BP algorithm is not guaranteed to achieve the best performance.

Recently, by adopting a data-driven strategy, the neural BP decoding algorithm [14] provides a potential solution to improve the performance beyond classical BP. As shown in Fig. 1, the neural BP method is essentially inspired by the "Deep Unfolding" [26] methodology. To be specific, it unfolds the Tanner graph of channel codes to a Trellis graph, which can be then interpreted as a neural network. Here the neurons in the odd and even layers represent $u_{v\to c}$ and $u_{c\to v}$, respectively. In other words, the neural BP decoder views the decoding process as a forward propagation of neural network. From this perspective, the connections between $u_{v\to c}$ and $u_{c\to v}$ are the trainable weights instead of the fixed 1 or 0 – a type of relaxation of the original "connected or not" in the classical BP decoder. Accordingly, the LLR update principles are changed as follows:

$$u^t_{v\to c} = f(w^t_{v,in} l_v + \sum_{c' \in N(v)\backslash c} w^t_{c'\to v} u^{t-1}_{c'\to v}), \tag{5}$$

$$u^t_{c\to v} = g(\prod_{v' \in M(c)\backslash v} u^t_{v'\to c}), \tag{6}$$

$$s_v^t = \sigma(w_{v,out}^t l_v + \sum_{c' \in N(v)} w_{c' \to v}^t u_{c' \to v}^t), \tag{7}$$

where $f(\cdot)$, $g(\cdot)$ and $\sigma(\cdot)$ denote the tanh, arctanh and sigmoid function, respectively. Here the trainable weights $w_{v,in}^t, w_{c' \to v}^t, w_{v,out}^t$ and $w_{c' \to v}^t$ can be optimized by minimizing the cross entropy(CE) loss as:

$$loss = \sum_{v=1}^{N} -[r_v \log s_v + (1 - r_v) \log(1 - s_v)]. \tag{8}$$

III. Review of Doubly Residual Neural Decoder

In this section we briefly review our recently proposed doubly residual neural (DRN) decoder [24], a type of new neural BP approach that can significantly improve the error-correcting performance with reduced decoding latency. More details can be referred to [24].

The key idea of DRN is to impose the famous residual block [27] on the structure of neural BP decoder design. This critical change is motivated by the grand success of ResNet: the use of residual block can efficiently mitigate the notorious vanishing gradient problems in deep neural networks. As indicated in Fig. 1, the unfolded neural BP decoders are typically very deep – their depths are twice the number of the iterations. So a simple feedforward architecture, highly probable, would suffer a vanishing gradient problem and thereby fail to achieve very high error-correcting performance. Therefore, a residual structured neural BP decoder, inherently, can be trained in a more stable way and potentially exhibit better BER performance.

Unlike the classical ResNet that is equipped with one type of residual connection, the DRN decoder used in channel coding contains two types of residual operations. First, the residual between s_v and $u_{c \to v}^t$ is calculated and serves as the input of the residual block:

$$a_{cv} = s_v - u_{c \to v}^t. \tag{9}$$

Notice that here we do not include the information carried by l_v. This is because as indicated in Eq. (7), s_v contains the most up-to-date LLR information than l_v during the entire iterative process. Therefore, merging l_v into s_v will not impact the overall decoding performance with the use of residual learning.

The second residual operation exists in the building block. Similar to the mechanism used in the ResNet, a_{cv} is input to a residual block and is combined with trainable weights w to generate the output as below:

$$b_{cv} = a_{cv} + h(w, a_{cv'}), \tag{10}$$

where $h(\cdot)$ denotes the activation function as the combination of $f(\cdot)$ and $g(\cdot)$ in Eq. (5) and Eq. (6). Overall, using the two types of residual operation (residual input and residual learning), the trainable weights w can be properly optimized with the standard backward propagation. In addition, to further reduce the computational cost, the *min-sum approximation* [28] can be used to simplify the calculation of tanh and arctanh as:

$$\begin{aligned} y &= 2 \operatorname{arctanh}[\tanh(\frac{p}{2}) \tanh(\frac{q}{2})] \\ &\approx \operatorname{sign}(p) \cdot \operatorname{sign}(q) \cdot \min(|p|, |q|). \end{aligned} \tag{11}$$

And hence $h(\cdot)$ can be reformulated as below:

$$h(w, a_{cv}) = w \min_{v' \in M(c) \backslash v} |a_{cv'}| \prod_{v' \in M(c) \backslash v} \operatorname{sign}(a_{cv'}). \tag{12}$$

Table I lists the negative logarithm of BER performance among different decoding methods. A higher number that corresponds to a lower BER denotes better performance. It is seen that for different LDPC codes, the built-in doubly residual structure enables our DRN algorithm to obtain better error-correcting capability than the classical BP algorithm and the state-of-the-art neural BP algorithm. Notice that here for a fair comparison, 5 iterations are set for all the listed algorithms.

IV. Hardware Architecture of DRN Decoder

A. Analysis of Design Strategy

In this section, we explore the efficient hardware architecture for DRN decoder. Different from the conventional design strategy for channel decoder hardware, here we propose to adopt systolic array, the commonly used architecture option for AI accelerator, to build the DRN decoder hardware. This choice is based on two considerations. First, systolic array is a very mature technique and its effectiveness has been widely demonstrated in several silicon products. Its inherent high data throughput and sufficient data reuse are very attractive for DNN-included hardware implementations. Second, systolic array inherently provides high reconfigurability to different DNN workloads. Considering the modern communication systems typically use various types of channel codes with different code-lengths and code rates, the high reconfigurability of the underlying hardware is very important and useful for practical deployment. Beyond that, because currently many other DSP modules can now also be re-designed using DNNs [29], [30], choosing the general systolic array architecture brings the future potentials of unifying the entire baseband processing hardware.

A major challenge of using systolic array for implementing DRN decoder is the sparsity issue. Though systolic array can perfectly execute dense matrix multiplication, its efficiency will be significantly degraded when processing sparse matrix multiplication, which is the component operation in the DRN decoder. Therefore, how to properly processing sparse computation should be carefully considered.

Next we have a detailed analysis of the computing procedure of DRN decoder. Based on the description in Section III, the Tanner graph of an example LDPC codes in Fig. 1 can be now unfolded to a doubly residual neural network as shown in Fig. 2. Here the 6 dotted-line neurons in the first column

Decoder	Conventional BP			Neural BP			DRN (Ours)		
SNR (dB)	4	5	6	4	5	6	4	5	6
LDPC (49, 24)	5.36	7.26	10.03	5.29	7.67	10.27	**5.77**	**7.86**	**11.28**
LDPC (121, 60)	4.76	7.20	11.07	4.96	8.00	12.35	**5.26**	**8.37**	**13.20**
LDPC (121, 70)	5.85	8.93	13.75	6.43	9.53	13.83	**6.39**	**10.10**	**15.43**
LDPC (121, 80)	6.54	9.64	14.78	7.04	10.56	14.97	**7.31**	**11.24**	**17.00**

TABLE I

NEGATIVE LOGARITHM OF BER PERFORMANCE OF DIFFERENT CHANNEL DECODING ALGORITHMS.

correspond to the 6 variable nodes in s_v. And the 12 edges in the Tanner graph correspond to the 12 black neurons in each layer. The initial $u^t_{c\to v}$ corresponds to the 12 green neurons in the first layer. Starting from the second layer, the green and black lines have the fixed weight as -1 and 1, respectively. In addition, the $minsum(\cdot)$ operation is denoted with blue lines starting at the third layer. Since in this example $d_c = 3$, so for each calculation process, we choose 3 neurons in s'_v as matrix A and 3 neurons in $u^t_{c\to v}$ as matrix B as indicated by the blue dotted-line rectangle in Fig. 2. To be specific, for A = $\begin{bmatrix} a1 & a2 & a3 \end{bmatrix}$ and B = $\begin{bmatrix} b1 & b2 & b3 \end{bmatrix}$ we have:

$$\begin{cases} b1 = minsum(a2, a3), \\ b2 = minsum(a1, a3), \\ b3 = minsum(a1, a2). \end{cases} \tag{13}$$

Regarding the processing schedule, for each layer the blue line-connected neurons that are involved with processing $u^t_{c\to v}(f(x))$ are updated; while the other neurons simply receive the values through shortcut from the previous layer (indicated by the curved arrow). To decide which neurons to be updated, the edges in the Tanner graph are divided into different groups, and the edges in the same column belong to the same group because they are referred to the same variable node in s_v. Since during each A to B computation there is only one edge involved in each group but the updated value should be applied to the whole group (see Eq. 10), multiple times of sub-update are needed to form an entire iteration of the DRN decoding. For instance, in this example the number of rows of $\mathbf{H}$ is 4, so 4 times of sub-update are required as shown in Fig. 2.

B. Hardware Architecture

Fig. 3 shows the corresponding overall hardware architecture of DRN decoder, which contains three main components: memory(SRAM and buffers), controller and computation module. Next we describe them in details.

a) Memory: Because the above discussed update process requires frequent memory access and multiple neurons need to be updated after each calculation, the weight and input data are initially stored in the SRAM and then they are outputted into the weight buffer and input buffer at the beginning of the neural network inference phase, respectively. s'_v is first temporarily stored in the input buffer and it is then outputted back to the SRAM once the decoding process ends. By adopting this strategy we can reduce unnecessary SRAM access and update the neurons simultaneously.

b) Transform Matrix Buffer: An individual transform matrix buffer is used to store the transform matrix illustrated in Eq. 13. In the example raised in Eq. 13, the transform matrix $T = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$. Notice that here T is a special $d_c \times d_c$-size symmetric matrix whose diagonal entries are 0 and all the other entries are 1. It is initialized by the main controller and then it is stored in the transform matrix buffer in a column-wise way.

c) Systolic Array: An one-dimension systolic array is adopted in this case study. Here array size is set as $d_c \times 1$ to perform the computation of Eq. 13 in each clock cycle. During the execution, the systolic array receives the serial input data x_i from the input buffer, the broadcast weight data w_i and transform matrix data t_i from the weight buffer and the transform matrix buffer, respectively. The dataflow is designed in an output stationary style, which means the output values $u^t_{c\to v}$ are stored in the processing elements (PEs). By adopting this strategy, the systolic array is essentially in charge of the computation of the dense part of each layer instead of the entire sparse layer, thereby avoiding the potential performance degradation.

d) Processing Element: The inner architecture of PE is shown in Fig. 4. Here x is the input from the input buffer and it goes through D_1 register to be sent to the adjacent PE. Meanwhile, its absolute value $|x|$ is extracted and feed into a comparator which is enabled by control signal. In this way the partial result of $minsum(\cdot)$ can be stored in D_2 register only when needed. When the update process starts, the partial result is output from D_2 to the multiplier, which performs the multiplication with w stored in the weight buffer. The product is then sent input to an XOR operator.

e) Routing Network: After the computation in the systolic array finishes, the temporary result $u_{c\to v}$ is simultaneously output from the PEs into the routing network. Through the routing network, the $u_{c\to v}$ are added up and the final results are sent into the input buffer to finish the update process. Fig. 5 shows part of the routing network. Here the routing network itself consists of multiple sets of adders that are directly attached to the registers in the input buffers. One set of adders corresponds to one group of edges. In this way we can achieve simultaneous update for multiple neurons.

V. FPGA IMPLEMENTATION RESULT

Based on the above described hardware architecture, we implement an example 16-bit FPGA prototype on Xilinx PYNQ-Z1 board . Here the chosen LDPC code has a parity

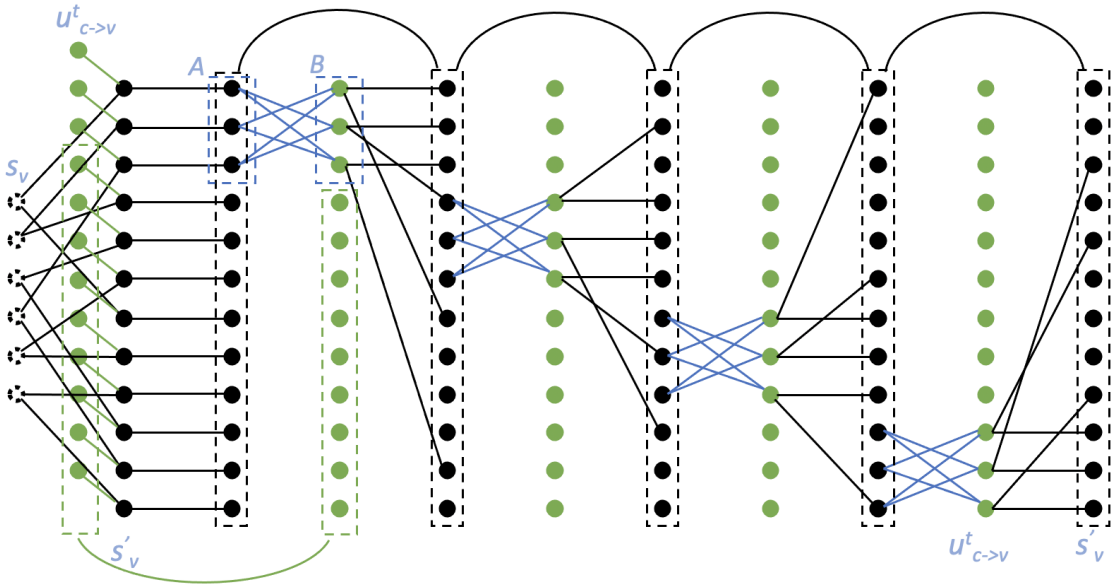

Fig. 2. Each iteration of DRN consists of multiple sub-update in the format of neural network.

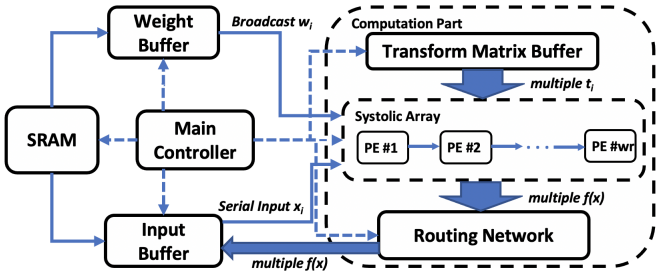

Fig. 3. Overall hardware architecture of DRN decoder.

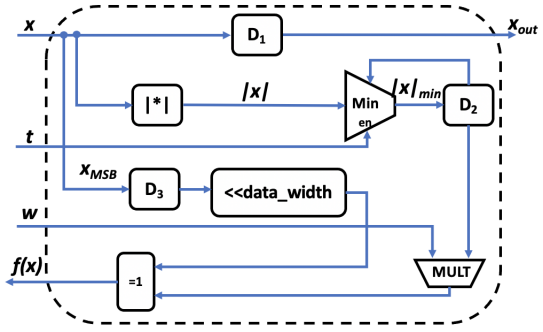

Fig. 4. The internal architecture of processing element.

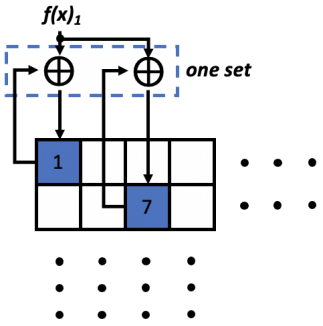

Fig. 5. Part of routing network.

	Utilization	Available	Utilization %
LUT	5600	53200	10.52
FF	7600	106400	7.15
DSP	11	220	5

TABLE II
FPGA UTILIZATION RESULT ON XILINX PYNQ-Z1

check matrix 55×121-size H with $d_c = 11$ and $d_v = 5$. In our design each PE contains one DSP to perform the multiplication so totally 11 PEs are allocated in the architecture. Since the maximum iteration is set as 5 iterations and $H_r = 55$, so there are 275 weights in the DRN decoder in total. Accordingly, the weight buffer has the size of 550B and the input buffer is 1.2KB. So the overall required SRAM size is about 2KB. Table II shows the resource utilization results. Here the total on-chip power consumption is 0.14W under the clock frequency as 237MHz.

VI. Conclusion

This paper explores the algorithm and hardware co-design for deep learning-powered channel decoder. We first revisit our recently proposed doubly residual neural (DRN) decoder to achieve very strong error-correcting performance. Then, a regular systolic array based hardware architecture is developed to accelerate the processing of the modified decoding algorithm. An example FPGA implementation is also developed to support the execution of DRN decoder on short LDPC codes.

References

[1] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2. IEEE, 1993, pp. 1064–1070.

[4] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.

[5] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.*, vol. 3, pp. 37–46, 1955.

[6] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 341–346.

[7] S. Cammerer, T. Gruber, J. Hoydis, and S. Ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[8] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2017, pp. 1–6.

[9] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 1361–1365.

[10] W. Xu, Z. Wu, Y.-L. Ueng, X. You, and C. Zhang, "Improved polar decoder based on deep learning," in *2017 IEEE International workshop on signal processing systems (SiPS)*. IEEE, 2017, pp. 1–6.

[11] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," *arXiv preprint arXiv:1805.09317*, 2018.

[12] W. Xu, X. You, C. Zhang, and Y. Be'ery, "Polar decoding on sparse graphs with deep learning," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2018, pp. 599–603.

[13] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes-a syndrome-based approach," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1595–1599.

[14] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[15] Y. Jiang, S. Kannan, H. Kim, S. Oh, H. Asnani, and P. Viswanath, "Deepturbo: Deep turbo decoder," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2019, pp. 1–5.

[16] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," in *Advances in Neural Information Processing Systems*, 2019, pp. 2329–2339.

[17] C. Deng and S. L. B. Yuan, "Reduced-complexity deep neural network-aided channel code decoder: A case study for bch decoder," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 1468–1472.

[18] B. Yuan, Z. Wang, L. Li, M. Gao, J. Sha, and C. Zhang, "Area-efficient reed-solomon decoder design for optical communications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 6, pp. 469–473, 2009.

[19] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 4, pp. 1241–1254, 2013.

[20] ——, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE transactions on signal processing*, vol. 62, no. 24, pp. 6496–6506, 2014.

[21] ——, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2268–2280, 2014.

[22] ——, "Llr-based successive-cancellation list decoder for polar codes with multibit decision," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 1, pp. 21–25, 2016.

[23] L. Li, B. Yuan, Z. Wang, J. Sha, H. Pan, and W. Zheng, "Unified architecture for reed-solomon decoder combined with burst-error correction," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 20, no. 7, pp. 1346–1350, 2011.

[24] S. Liao, C. Deng, M. Yin, and B. Yuan, "Doubly residual neural decoder: Towards low-complexity high-performance channel decoding," *arXiv preprint arXiv:2102.03959*, 2021.

[25] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[26] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," *arXiv preprint arXiv:1409.2574*, 2014.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[28] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding ldpc codes," in *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No. 01CH37270)*, vol. 2. IEEE, 2001, pp. 1036–1036E.

[29] S. Liao, C. Deng, L. Liu, and B. Yuan, "Structured neural network with low complexity for mimo detection," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2019, pp. 290–295.

[30] S. Liao, C. Deng, Y. Xie, L. Liu, and B. Yuan, "Low-complexity neural network-based mimo detector using permuted diagonal matrix," in *2020 54th Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2020, pp. 111–115.