A Low-Latency Syndrome-based Deep Learning Decoder Architecture and its FPGA Implementation

E. Kavvousanos and V. Paliouras
Electrical and Computer Engineering Department
University of Patras, Greece

***Abstract*—Recently, Machine Learning has been considered as an alternative design paradigm for various communications subsystems. However, the works that have assessed the performance of these methods beyond the algorithmic level are limited. In this paper, we implement in hardware and evaluate the performance of the Syndrome-based Deep Learning Decoder for a BCH(63,45) code in terms of throughput rate and latency. The implemented Neural Network is compressed by applying pruning, clustering and quantization to an 8-bit fixed-point representation, with no significant loss in its BER performance, while achieving 90% weight sparsity in each layer. An FPGA architecture is designed for the decoder which exploits the compressed structure of the Neural Network in order to accelerate the underlying computations with moderate hardware requirements. Experimental results are provided which show that the decoder achieves latency less than a tenth of a millisecond and a throughput rate up to 5 Mbps, substantially outperforming previous implementations by $30\times$.**

I. Introduction

Machine Learning (ML) has seen impressive research accomplishments in the last decade. Deep Learning (DL) and Artificial Neural Networks (ANN) have gained extensive attention and have seen great adoption in manifold data-driven problems and applications. The extent of the research effort is justified considering the unparalleled accuracy that various ANN architectures exhibit in quite diverse problems.

Recent research works in communication systems follow a data-driven approach by utilizing Machine Learning methods to address several open problems in the Physical and Medium Access Control layers. These new ML-oriented techniques are applied in different components of communication systems individually, e.g., channel coding and decoding, channel estimation, or to jointly optimize the transmitter and the receiver.

Regarding Error Correction, decoders have been introduced for binary linear block codes which approach maximum-likelihood performance for small block lengths. Nachmani *et al.* [1] build a Neural Network based on the Belief Propagation (BP) algorithm and the associated Tanner Graph which improves the decoding performance for small dense codes. Xu *et al.* [2] also build a BP-based ANN Decoder for Polar Codes and propose a hardware implementation. Teng *et al.* [3] exploit the iterative nature of the BP-NN for Polar Codes and apply weight sharing and quantization between iterations. Bennatan *et al.* [4] exploit the syndrome and the reliability of the received block to extract the error pattern by utilizing a trained ANN. This Syndrome-based DL Decoder (SDLD) can be considered as a universal decoding method as the ANN can be trained for any code and channel model. SDLD is also studied in [5], where a hardware implementation is showcased using generic DL accelerators, so-called DPUs, in FPGA.

ANNs owe their impressive accuracy to intensive computations. Modern ANN architectures require millions of multiply-accumulate (MAC) operations and excessive data movement and storage imposing limitations to high performance, low power and resource utilization rendering their implementation a challenge. Han *et al.* propose an ANN compression methodology which includes weight pruning, clustering and quantization. The compressed ANN significantly reduces the required MACs and memory with minimal loss of accuracy. Special hardware can be designed that take advantage of the imposed compression, accelerating ANN inference [6],[7].

While DL-enhanced decoders are limited to short block lengths, they still can be of interest in applications where low latency and high reliability are needed. For example, the Ultra-Reliable Low Latency Communications (URLLC) feature of 5G [8] poses such requirements.

In this paper, we design an SDLD architecture suitable for implementation on an FPGA device. To accelerate the involved ANN, a compression method is applied by pruning, clustering and quantizing its weights. Furthermore, the inference of the ANN is accelerated by introducing a Sparse Matrix-Vector Multiplication processor which exploits both weight matrix and input vector sparsities. Moreover, the weight matrix of each layer is partitioned for better parallelism. All the required parameters are stored in on-device FPGA memories to avoid external DRAM accesses. The layers of the ANN are operated in a pipelined fashion to maximize throughput rate. To the best of our knowledge, this paper showcases a novel DL decoder architecture and FPGA implementation that leverages both weight and layer input sparsity to achieve $30\times$ higher throughput rate than prior art. The measurements and verification prove the feasibility of the SDLD implementations with moderate hardware requirements; hardware complexity *vs.* throughput rate and latency trade-offs are investigated, demonstrating the potential of this type of decoder.

The remainder of this paper is organized as follows: Section II gives an overview of the SDLD. Section III describes

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH-CREATE-INNOVATE (project code: T1EDK-02551).

the methodology for the compression of the employed ANN. Section IV presents the decoder architecture designed for FPGA implementation. Section V evaluates the implementation and compares it to alternative decoding methods. Finally, Section VI discusses our findings and concludes the paper.

II. Overview of the Syndrome-Based DL Decoder

A. Notation

Let a binary information message of k bits $\boldsymbol{m}_b \in \{0,1\}^k$, be encoded to a codeword $\boldsymbol{x}_b \in \mathcal{C}$, where $\mathcal{C} \subset \{0,1\}^N$ is the codebook of the (N,k) binary linear block code. We assume Binary Phase-Shift Keying (BPSK) modulation, therefore the bits of the codeword $\boldsymbol{x}_b$ are modulated to $\boldsymbol{x}_s$, via the mapping $x_{s,i} = 1 - 2 \cdot x_{b,i}$. We note that the subscripts b and s are used to denote binary (*i.e.*, $\{0,1\}$) and bipolar (*i.e.*, $\{-1,1\}$) objects respectively. The modulated symbols are transmitted over the channel and the receiver obtains a sequence $\boldsymbol{y} \in \mathbb{R}^N$. The sequence $\boldsymbol{y}$ is related to $\boldsymbol{x}_s$ via the channel noise statistics. In the case of AWGN channel, the received sequence is described by $\boldsymbol{y} = \boldsymbol{x}_s + \boldsymbol{n}$, where $\boldsymbol{n}$ is the channel noise vector.

B. Syndrome-Based DL Decoder Operation

SDLD [4] relies on the estimation of the noise pattern that distorts the received word. Decoding consists of three stages; pre-processing, noise estimation and post-processing.

In the pre-processing stage, the decoder receives the channel output $\boldsymbol{y}$ (reliabilities) and extracts the bipolar hard decision $\boldsymbol{y}_s = \operatorname{sign}(\boldsymbol{y})$ and the absolute values $|\boldsymbol{y}|$. The binary hard decision of the channel output is also computed as $\boldsymbol{y}_b = 0.5 \cdot (1 - \boldsymbol{y}_s)$. Consequently, $\boldsymbol{y}_b$ is used along with the $(N-k) \times N$ parity check matrix $\boldsymbol{H}$ of the code, in order to generate the syndrome of the received sequence, $\boldsymbol{s} = \boldsymbol{H} \cdot \boldsymbol{y}_b$.

The noise-estimation stage is the core computational module of the decoder and is implemented as an ANN that has been trained to identify the erroneous bits in the received sequence. Its operation is described by:

$$\boldsymbol{z} = f_L(f_{L-1}(\dots f_l(\dots f_1(\boldsymbol{v})))), \tag{1}$$

where $\boldsymbol{v} = [\boldsymbol{s}, |\boldsymbol{y}|]$ is the input to the noise-estimation ANN and comprises the syndrome along with the absolute values of the channel reliabilities. Its output $\boldsymbol{z}$ is an N-dimensional vector, which corresponds to the estimated noise pattern. Here, $f_l(\boldsymbol{u}_l; \boldsymbol{\theta}_l)$, $l = 1, 2, \dots, L$ resembles to a non-linear transformation, *i.e.*, layer, of the ANN with its respective parameters $\boldsymbol{\theta}_l$. The output layer $f_L(\boldsymbol{u}_L; \boldsymbol{\theta}_L)$ uses a $\tanh$ or a sigmoid function to classify each bit. The depth of the network L and the size of each layer can be selected arbitrarily, trading error-correcting accuracy with computational complexity.

Each output element $\boldsymbol{z}_i, \ i = 1, 2, \dots N$ corresponds to the respective bit in the codeword. For the case of $\tanh$ activation in the output layer, if $z_i > 0$ then the respective bit in the sequence is considered correct, else if $z_i < 0$ then the respective bit is in error. The magnitude of $|\boldsymbol{z}_i|$ indicates the likelihood of the i-th bit being correct or erroneous. Finally, the estimated error pattern from the ANN is combined with the hard decisions $\hat{\boldsymbol{x}}_s = \boldsymbol{y}_s \cdot \operatorname{sign}(\boldsymbol{z})$ in the post-processing stage in order to flip the faulty bits.

III. Compression of the Noise Estimation ANN

A fully-connected (FC) layer is a common transformation used in modern ANNs. Essentially, it incorporates a matrix-vector multiplication (MVM) of the weight matrix $\boldsymbol{W}$ and the input vector $\boldsymbol{u}$, the addition of a bias vector $\boldsymbol{b}$ (which can be fused in $\boldsymbol{W}$) and the application of a non-linear function g, *i.e.*

$$\boldsymbol{\phi} = g(\boldsymbol{W} \cdot \boldsymbol{u} + \boldsymbol{b}). \tag{2}$$

The MVM dominates the computational load in an FC layer. In many ANNs the weight matrix $\boldsymbol{W}$ may have hundreds of thousands or even millions of parameters which translate to an equal amount of multiply-accumulate operations.

In this work we implement in FPGA an SDLD trained for the case of the BCH(63,45) code. For the Noise Estimation ANN an organization of seven fully-connected layers is chosen as in [5]. More complex organizations can potentially lead to better error correcting capability trading off computational complexity. The ANN accepts an input vector of length $N + (N - K) = 81$. For the hidden layers, the output size is set to 300 and the Rectified Linear Unit (RELU) activation is used. The activation of the output layer is $\tanh$. The total number of parameters used by this architecture is $495,063$.

The ANN is trained with the ADAM Stochastic Gradient Descent variant [9] using mini-batches of $8,196$ samples per training step. The samples are generated assuming an AWGN channel with a noise level of $E_b/N_0 = 4$ dB. The ANN is then validated for various noise levels to assess its generalization and error correcting capability.

Having a trained baseline model, compression methods are applied for the Noise Estimation ANN to deploy the SDLD in FPGA efficiently with minimal loss in accuracy. Initially, we apply gradual magnitude-based pruning [10] and eliminate low-magnitude weights of the FC layers while fine-tuning for a few epochs in order for the network to recover in between pruning steps. Furthermore, the weight matrices are partitioned into several submatrices according to the number of Processing Elements that are to be used for the acceleration of the underlying computations as described in Section IV-A. All submatrices are pruned to the same sparsity level in order to balance the MACs across the computation resources [7],[11]. This way, an overall 90% sparsity is achieved for every layer. Then, clustering is applied independently to the remaining weights of each layer using the K-means algorithm. For each layer 64 clusters are used and the cluster centroids are quantized to 8 bits fixed-point. Thus, Eq. (2) becomes

$$\phi_i = g\left(b_i + \sum_{\{j | w_{i,j} \neq 0\}} C_{I_{i,j}} \cdot u_j\right) \tag{3}$$

for the i-th element of the activation vector. Here C and $I_{i,j}$ denote the cluster centroids look-up table and the cluster index of the non-zero weight $w_{i,j}$ respectively.

IV. Proposed FPGA Architecture for the Decoder

Following the algorithmic simplifications of Section III, we design an SDLD hardware architecture for the BCH(63,45) code, exploiting any parallelism possible to achieve low latency and high throughput rate. We use Xilinx's Vitis HLS

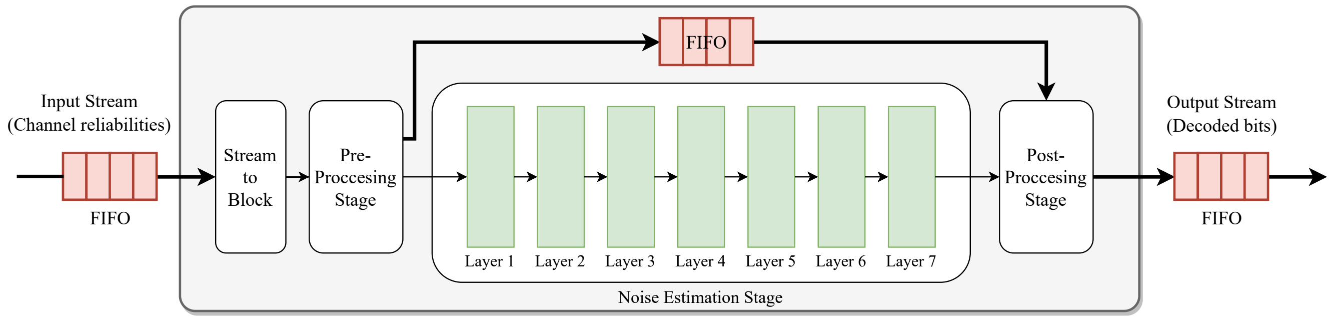

Fig. 1. Block Diagram of the Decoder FPGA Architecture.

for rapid development and architecture space exploration and model the system in High Level Synthesis C++.

Fig. 1 shows the overall architecture of the decoder. The decoder receives a continuous stream of channel reliabilities from a FIFO and packs vectors with length of $N = 63$. The reliability vector is pre-processed and input vector for the Noise Estimation ANN is derived. The binary hard decisions $\boldsymbol{y}_b$ are forwarded to an N-bit wide FIFO for the post-processing stage. The elements of the ANN input vector are quantized to 8-bit fixed-point representation with 3 integral and 5 fractional bits (FXP-3.5). The output of the ANN is in FXP-1.7 representation. The post-processing stage receives the output of the ANN along with the binary word from the FIFO and flips the erroneous bits. The corrected block is forwarded to an outbound FIFO. All modules in Fig. 1 operate concurrently in a dataflow pipeline. The data transfers between the modules are performed via Ping-Pong buffers or FIFOs.

A. Acceleration of the Sparsely-Connected Layer

In order to maximize the throughput rate and minimize the latency of the decoder the FC layers of the Noise Estimation stage are accelerated by exploiting the compression described in Section III. The weight matrix in every layer is sparse, with 10% non-zero values. We exploit this sparsity by designing a Sparse Matrix-Vector Multiplication (SPMVM) accelerator.

The general organization of the Sparsely-Connected (SC) Layer architecture is illustrated in Fig. 2. The layer consists of an arbitrary number of Processing Elements (PE) which execute the multiply-accumulate (MAC) operations involved in the SPMVM. In order to exploit parallelism, the operations of each PE correspond to a set of rows in the weight matrix following a cyclic manner. For example, assuming P PEs, the p-th PE, where $p = 0, 1, \ldots, P-1$, calculates MACs associated with weights $w_{i,j}$ for which $i \mod P = p$. Furthermore, the weight matrix is partitioned with the same principle and each PE is given a private weight memory locally by using FPGA Block RAMs and LUTs. The local, on-device, storage of the weights is feasible due to the relatively small size of the ANN and the pruning applied. Each of the P sparse matrices are stored in Compressed Sparse Column (CSC) format. Since clustering is applied to the weights, we save for each non-zero weight the index to its container cluster. As we use 64 clusters, 6 bits are required by each index. Each PE decodes the respective weight using the index and a private look-up table with the cluster centroids, quantized to FXP-1.7.

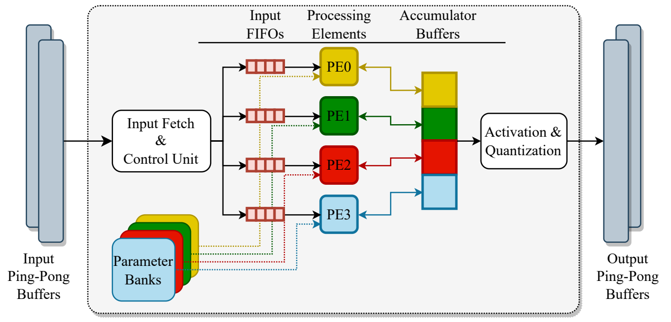

Fig. 2. Block Diagram of the implemented Sparsely-Connected Layer, for the case of 4 PEs.

The CSC format is preferable to alternatives because when the MVM is calculated column-wise, we can take advantage of input sparsity as well, in addition to the weight sparsity. Since RELU is used as the activation of the hidden layers, a considerable number of zeros is expected to the input of subsequent layers. Thus, exploiting the particular zeros, MACs are skipped in columns which have zero input value. Our measurements reveal that layer activations contain about 50% zeros, a percentage that slightly varies between layers.

The *Input fetch and Control Unit* forwards 8-bit input values to the *Input FIFOs* of the PEs and manages their operation. The PEs update the *Accumulator Buffer* which stores the partial product sums. The precision of the MACs is kept in FXP-10.22. Finally, the activation is applied to the final product sums and the output is quantized with 8 bits. The fixed-point representation of the output is determined by the dynamic range requirements, calculated for every layer off-line by feeding the ANN with an evaluation dataset. The activation of the output layer is approximated by a hard $\tanh$ function.

V. Experimental Results

A. Hardware Implementation Results

To implement the hardware prototype of the decoder, we use the ZCU104 evaluation board from Xilinx, which accommodates a ZU7EV MPSoC device. The verification setup includes a Direct Memory Access (DMA) system which streams data in and out of the decoder. Software running on the ARM CPU of the device generates testing data, saves it in the board DRAM and then instructs the DMA to continuously stream the data to the decoder. Also, the DMA is instructed to transfer the decoded data stream to the DRAM in order for the CPU software to measure the decoding performance.

The error-correcting capability of the implemented decoder

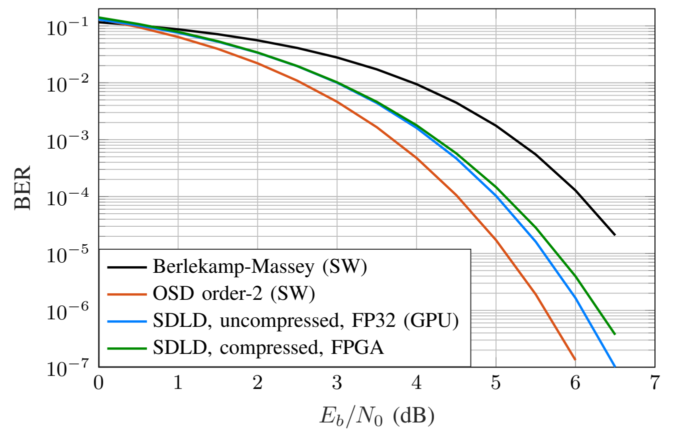

Fig. 3. BER performance of the implemented decoder and comparison with software (SW) alternatives, for BCH(63,45).

TABLE I
DECODER PERFORMANCE

PEs/Layer	Latency (μs)	Throughput rate (Mbps)
4	154	2.7
8	111	3.7
16	83	5.0

is visualized in Fig. 3 as a Bit Error Rate (BER) *vs.* normalized Signal-to-Noise Ratio (E_b/N_0) plot. The BER performance of the proposed implementation is very close to that of the initial non-compressed FP32 model with an expected degradation due to the applied compression. The BER performance of Berlekamp-Massey and the Ordered Statistics Decoder (OSD) [12] for BCH(63,45) are also shown for comparison. The performance of the OSD order-2 approaches maximum-likelihood decoding. By employing a more complex ANN it is possible to approach maximum-likelihood decoding by trading off decoding latency and hardware resources.

In order to investigate hardware complexity *vs.* processing performance trade-offs in the proposed architecture, we measure the latency and throughput rate as a function of the number of PEs per SC layer, for various implemented instances of the prototype. In all cases, we use a system clock of 200 MHz. Table I reports the achieved measured latency and throughput rate for each case. It can be observed that by using more PEs per layer, the decoder achieves an anticipated speed-up, at the cost of more resources, as seen in Table II.

B. Comparison

We compare the measurements of the implemented SDLD with other DL-based decoding methods. Despite that several DL-enabled decoders have been reported in the literature, hardware implementations are scarce. Furthermore, different codes, decoding algorithms, and technologies are employed. Considering the BP-based Polar NND [2], its latency for a Polar code of length $N = 64$ is given by $101 \cdot t_p$, where t_p is the processing time of each stage. This decoder targets specific codes as the underlying network exploits the corresponding BP graph. If t_p is comparable to the latency of an SC layer in our architecture, then our implementation may experience less latency overall due to fewer layers needed. It would be interesting to see a hardware implementation of the Polar NND in the future and compare experimental measurements.

TABLE II
FPGA RESOURCES COUNT AND DEVICE UTILIZATION

PEs/layer	Resources			
	CLB	LUT	Registers	BRAM
4	5147 17.87%	29837 12.95%	7815 1.70%	18.5 5.93%
8	7546 26.20%	44593 19.35%	14546 3.16%	36.5 11.70%
16	11500 39.93%	70089 30.42%	25050 5.44%	58.5 18.75%

The SDLD implementation in [5] is substantially outperformed since its throughput rate is limited to 165 kbps. Also, its latency is considerably higher since an embedded CPU is involved for the control of the employed DPU accelerators.

VI. CONCLUSION

In this paper, we showcased the FPGA implementation of a Syndrome-Based Deep Learning Decoder. The underlying Noise Estimation ANN is compressed and an accelerator has been designed to fully exploit the 90% sparsity introduced to the ANN. We report a throughput rate of 5 Mbps and a latency of 83 μs for the case of 16 processing elements per ANN layer, substantially improving related prior art in terms of throughput. It follows that practical forward error correction hardware SDLDs are possible when both layer input and parameter sparsity are exploited.

REFERENCES

[1] E. Nachmani, E. Marciano *et al.*, "Deep Learning Methods for Improved Decoding of Linear Codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, 2018.

[2] W. Xu, Z. Wu *et al.*, "Improved polar decoder based on deep learning," *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, 2017.

[3] C.-F. Teng, C.-H. D. Wu *et al.*, "Low-complexity Recurrent Neural Network-based Polar Decoder with Weight Quantization Mechanism," *ICASSP 2019*.

[4] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep Learning for Decoding of Linear Codes - A Syndrome-Based Approach," *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018.

[5] E. Kavvousanos and V. Paliouras, "Hardware Implementation Aspects of a Syndrome-based Neural Network Decoder for BCH Codes," *2019 IEEE Nordic Circuits and Systems Conference (NORCAS)*, pp. 1–6, 2019.

[6] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," *CoRR*, vol. abs/1510.00149, 2015.

[7] S. Han, X. Liu *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 243–254.

[8] M. Shirvanimoghaddam, M. S. Mohammadi *et al.*, "Short block-length codes for ultra-reliable low latency communications," *IEEE Communications Magazine*, vol. 57, no. 2, pp. 130–137, 2019.

[9] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego*, Y. Bengio and Y. LeCun, Eds., 2015.

[10] M. Zhu and S. Gupta, "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression," *ArXiv*, vol. abs/1710.01878, 2018.

[11] S. Cao, C. Zhang *et al.*, "Efficient and Effective Sparse LSTM on FPGA with Bank-Balanced Sparsity," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. Association for Computing Machinery, 2019, p. 63–72.

[12] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, Sep 1995.