2024 Design, Automation & Test in Europe Conference (DATE 2024)

A Hardware Accelerated Autoencoder for RF Communication using Short-Time-Fourier-Transform Assisted Convolutional Neural Network

1st Kuchul Jung
Georgia Institute of Technology
Atlanta, US
kjung62@gatech.edu

2nd Jongseok Woo
Georgia Institute of Technology
Atlanta, US
jw@gatech.edu

Saibal Mukhopadhyay
Georgia Institute of Technology
Atlanta, US
saibal@

***Abstract*—This paper presents a hardware-accelerated autoencoder (AE) for wireless communication using a Short-Time-Fourier-Transform Assisted Convolutional Neural Network (STFT-CNN-AE). The design aims to reduce the autoencoder's resource requirements and power dissipation while maintaining its performance even in low Signal-to-Noise Ratio (SNR) wireless channels. The STFT-CNN-AE was implemented and tested on a Zynq UltraScale+ FPGA platform. Prototype measurements show that the STFT-CNN-AE achieves 3.5 times higher throughput at 2.6 times faster frequency, consumes 59% less power, and requires 76% fewer hardware resources (LUT and DSP) compared to a prior Multi-Layer Perceptron-based AE (MLP-AE). These improvements were achieved while maintaining comparable performance in low SNR (<7.5dB) channels.**

Keywords— *Autoencoder, CNN, STFT, HW Implementation*

I. Introduction

Noise and interference in wireless channels lead to signal degradation and increased Bit Error Rate (BER) in wireless communication. Several techniques have been proposed to address these issues, including channel coding, modulation, equalization, and diversity schemes. Recently, deep learning approaches have emerged as promising solutions, optimizing wireless communication systems through end-to-end learning [1-7]. These approaches aim to optimize transmitter, receiver, and channel models using extensive training data. An autoencoder (AE) based communication system, comprising an encoder in the transmitter and a decoder in the receiver (Fig. 1), has shown potential in facilitating the recovery of transmitted data in noisy environments [1-7].

Autoencoders can be employed to learn Channel State Information (CSI) from pilot symbols or training sequences and generate optimal transmitted signals to achieve high data rates and low BER. Designing autoencoders specifically for wireless communication, however, presents several challenges, including requirements for low-latency and energy-efficient implementations, limited availability of training data for diverse channel conditions, and the non-differentiable and non-linear nature of wireless channels. In conventional OFDM systems, a trainable constellation mapper is used in the transmitter, and a de-mapper in the receiver to compute the log-likelihood ratio (LLR) of the transmitted bits [3] (Fig. 2).

Recent advancements have focused on enhancing autoencoders using deep neural networks (DNNs) for the mapper and de-mapper stages [5-7]. These DNNs can be trained using stochastic gradient descent (SGD) and backpropagation, assuming differentiable channel models such as Additive White Gaussian Noise (AWGN) [6]. For example, Ji et al. [4] utilized convolutional layers and residual connections in DNN-based autoencoders, while Kim et al. [5] demonstrated an FPGA implementation of an autoencoder with a fully connected Multi-Layer Perceptron (MLP) architecture consisting of four layers (MLP-AE) (Fig. 3). However, using MLP layers risks increased computational

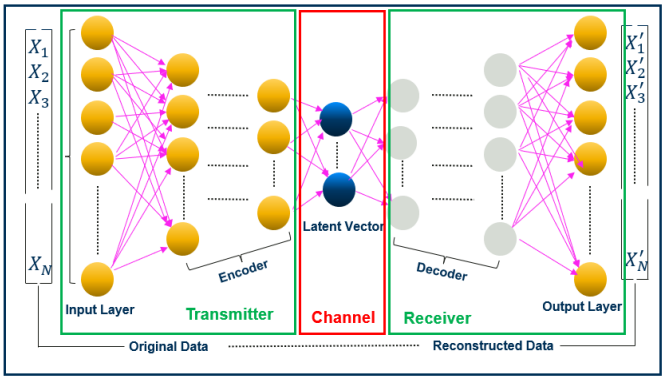

Fig 1. .Autoencoder-based Communication System

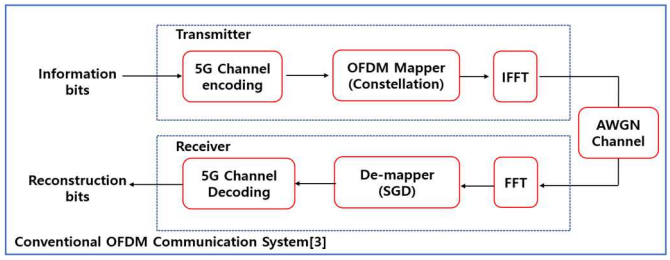

Fig 2. Conventional OFDM Communication System.

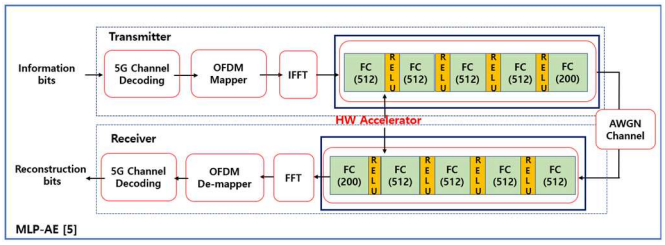

Fig 3. Communication System Architecture with MLP based Autoencoder (MLP-AE)

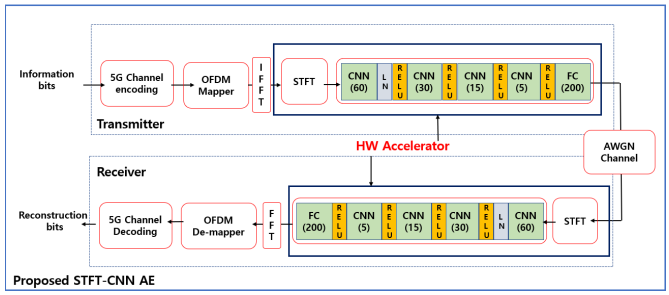

Fig 4. This Work: Communication system architecture with STFT-CNN based Autoencoder (STFT-CNN-AE).

complexity, and overall, prior works demonstrate poor performance in noisy channels and require large computational resources.

This paper presents a hardware-friendly, computationally efficient DNN-based autoencoder for wireless communication systems, using a CNN-based architecture optimized for high-throughput on-chip acceleration (Fig. 4). The model's performance is improved, and complexity is reduced through STFT-based pre-processing (STFT-CNN). A dedicated hardware accelerator (STFT-CNN-AE) is implemented and evaluated on a Zynq UltraScale+ FPGA board. Compared to MLP-AE, the proposed STFT-CNN-AE significantly reduces computation parameters by 79%, FPGA resource utilization by 76%, power consumption by 59%, and achieves 3.5 times higher throughput, achieving a more suitable solution for wireless communication systems that require low power and low latency while maintaining high performance.

II. Background

A. OFDM Wireless Communication System

OFDM is a prevalent modulation technique for wireless communication, dividing a wideband channel into orthogonal subcarriers to enable efficient frequency use and robustness against fading channels. Though commonly used in Wi-Fi, LTE, and 5G, OFDM encounters interference issues that require effective signal processing techniques to mitigate their impact on wireless communications.

B. Autoencoder for OFDM System

Autoencoders are neural networks used for unsupervised learning, aiming to minimize the discrepancy between input data and reconstructed output, as shown in Fig. 1. In wireless communication, they handle channel coding, modulation, and interference suppression. Their performance improves with deep learning by capturing intricate nonlinear mappings. Pairing autoencoders with OFDM systems enhances noise immunity, channel equalization, throughput, and reliability. We present two baseline autoencoder designs for comparison.

Conventional OFDM [Baseline 1]. In the traditional model, data bits from a random binary source experience 5G channel encoding and 64 QAM OFDM mapping. Post-IFFT processing, the signal passes through the channel. At the receiving end, the signal transitions back to the time domain via FFT, and then undergoes OFDM De-mapping and 5G channel decoding to retrieve the original signal. The recovered signal is assessed against the original, gauging its fidelity through the Bit Error Rate (BER) or the Block Error Rate (BLER).

Multi-Layer-Perceptron based AE (MLP-AE) [Baseline 2]. The second model we evaluated uses an MLP-based AE architecture, as depicted in Fig. 3. After undergoing IFFT at the transmitter, the MLP processes the signal before being channeled. On the receiver end, it first passes through the MLP, then is subjected to FFT for OFDM De-mapping to retrieve the initial signal. This MLP-based AE configuration consists of five layers: the initial four each having 512 neurons, and the final one having 200 neurons to align with the original signal's dimensions. By integrating the MLP architecture, we aim to enhance the Block Error Rate (BLER)

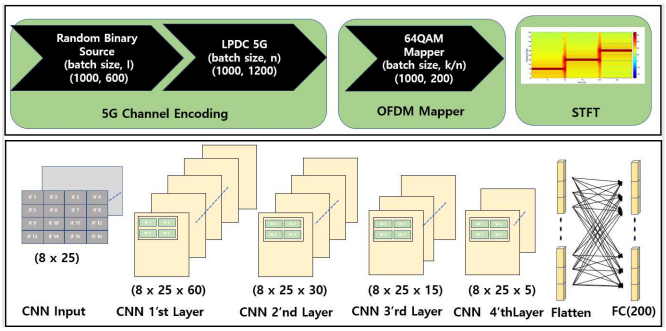

Fig 5. Data Flow of STFT-CNN-AE

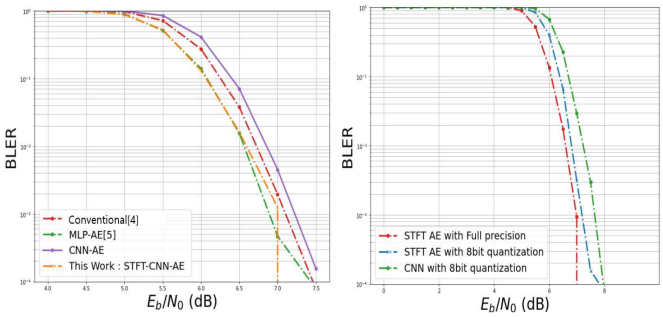

Fig 6. BLER Analysis: (top) comparison with prior works and (bottom) effect of precision for STFT-CNN-AE

by capitalizing on its prowess in detecting and illustrating intricate signal data correlations.

There is growing interest in developing hardware-friendly autoencoder architectures for real-time applications. Designing such architectures involves careful consideration of computational complexity, model accuracy, and hardware resource utilization.

C. Short-Time Fourier Transform (STFT)

Short-Time Fourier Transform (STFT) is a widely used signal processing technique that converts a time-domain signal into a frequency-domain representation. It works by dividing the signal into overlapping short-time segments, applying the Fourier transform to each segment, and then concatenating the resulting spectra to form a 2D time-frequency representation.

Mathematically, the STFT of a time-domain signal $\boldsymbol{f(t)}$ and a window function $\boldsymbol{\eta(t-t')}$ is defined as follows:

$$STFT(\tau,\omega) = \int_{-\infty}^{\infty} f(t)\eta(t-\tau)\, e^{-j\omega t} dt \quad (1)$$

where τ is the time shift parameter and ω is the frequency parameter. The output of the STFT is a 2D matrix of the signal's time-varying frequency components, which is commonly known as a spectrogram. The STFT can be used to analyze a wide range of signals, including audio, images, and RF signals

III. STFT-CNN-AE : Algorithm

In this segment, we outline the incorporation of the STFT algorithm into our autoencoder tailored for wireless communication systems. The STFT conducts Fourier Transforms on short, overlapping windows, yielding a time-frequency portrayal of the signal. This windowing and

overlap minimize spectral leakage and boost frequency clarity. Through STFT, quantization of CNN activations and weights is achievable. Coupled with sliding convolution, this lessens the model's intricacy. Our tests show that utilizing STFT for preprocessing amplifies autoencoder efficacy in noisy channel scenarios.

A. STFT-CNN Autoencoder

Fig. 4 illustrates our proposed STFT-Convolutional Neural Network (CNN) autoencoder design for wireless Orthogonal Frequency Division Multiplexing (OFDM) systems. This design focuses on boosting noise resilience and pertinent feature extraction. In the transmitter, data goes through 5G channel encoding, OFDM mapping, and Inverse Fast Fourier Transform (IFFT). The IFFT result then passes through STFT, emphasizing noise resistance and feature extraction. The STFT-CNN segment discerns both temporal and spectral attributes of the signal, pulling out vital features from its time-frequency depiction. After transmission through a channel introducing additive white Gaussian noise (AWGN), the noise-tainted received signal undergoes the STFT-CNN for denoising and feature extraction. The denoised features are transformed back to the time domain using the Fast Fourier Transform (FFT), followed by OFDM de-mapping and 5G channel decoding, thereby recovering the original signal. The proposed STFT-CNN-based autoencoder architecture offers several advantages, including enhanced noise immunity, effective feature extraction, and suitability for various communication scenarios characterized by noise and channel impairments.

B. Data Flow in STFT -CNN-AE

Fig. 5 presents the data flow of the STFT-CNN-AE. At the transmitter, the input is encoded using the 5G LDPC code, a robust channel encoding method that strikes a balance between performance and complexity. Beneficially decoded using message-passing algorithms, LDPC codes offer flexibility via puncturing and shortening. Standardized for 5G NR's data and control channels, they play a pivotal role in achieving high data rates, minimal latency, and dependable communication in upcoming wireless systems.

The OFDM mapper in the OFDM modulation places complex modulated symbols onto subcarriers in the frequency domain. In a 64QAM scheme, each symbol represents 6 bits, yielding 64 potential values. This mapper ensures symbols are correctly placed onto subcarriers by adjusting their phase and amplitude. Modulated I/Q samples are then passed through STFT and CNN layers within the transmitter. The input from the random binary source has a shape of (batch size, I), where the batch size is 1000 and I stands for 600 information bits. Post 5G LDPC Encoding, this shape evolves to (batch size, n), with 'n' being the codeword length determined by a code rate 0.5. The 64QAM de-mapper then reshapes this to (batch size, n/k), with 'k' denoting bits per symbol (6 for 64QAM), culminating in an STFT input shape of (1000, 200).

The STFT transformation, pivotal for assessing spectral properties of signals, is influenced by window size and overlap options, which determine frequency and time resolution, respectively. Through experimentation, optimal parameters were identified: a window size of 8 and no overlap. This configuration results in each signal having an STFT output shape of (8, 25). The spectrograms derived from the STFT serve as essential tools for analyzing frequency content and tracing patterns over time.

Table 1. Comparison of the Number of Model Parameters

MLP-AE[5]			STFT-CNN-AE		
Layer	Layer Type	#Model Parameters	Layer	Layer Type	#Model Parameters
1	Dense (512)	102,912	1	CNN (60)	300
2	Dense (512)	262,656	2	CNN (30)	7,230
3	Dense (512)	262,656	3	CNN (15)	1,815
4	Dense (512)	262,656	4	CNN (5)	305
5	Dense (200)	102,600	5	Dense (200)	200,200
Total		993,480	Total		209,850

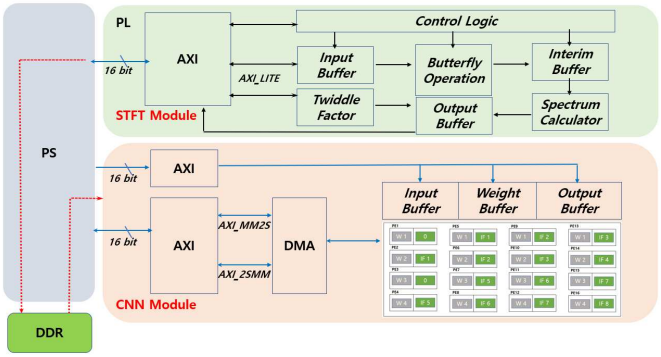

Fig 7. FPGA Architecture of Proposed STFT-CNN

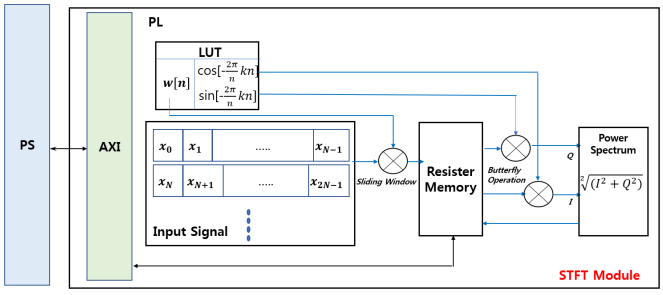

Fig 8. STFT Implementation

The designed wireless system integrates the IFFT with STFT-transformed data prior to transmission. While STFT heightens spectral resolution, IFFT incorporates OFDM for effective time-domain signal transmission. Together, they augment noise immunity, channel equalization, and overall system performance. Our system is modeled on an AWGN channel known for its noise-induced signal disruption and interference. Training the system across noise levels from 0 to 10 dB ensures adaptability in real-world settings.

Upon reception, the signal, marred by noise and channel distortions, is routed through a receiver mirroring the transmitter's architecture. This signal undergoes STFT processing and is then channeled through a CNN layer. It's next transformed back into the frequency domain via FFT. The culmination is the restoration of the original signal via OFDM de-mapping and 5G channel decoding.

Table 1 shows a notable reduction in model parameters across layers. For perspective, the MLP-AE consists of five interconnected layers, each with 512 nodes, rounded off by an output layer of 200 nodes. When counting model parameters, the MLP-AE and STFT-CNN-AE stand at 993,480 and 209,850, respectively. A noteworthy feature is our employment of STFT to curtail computational precision,

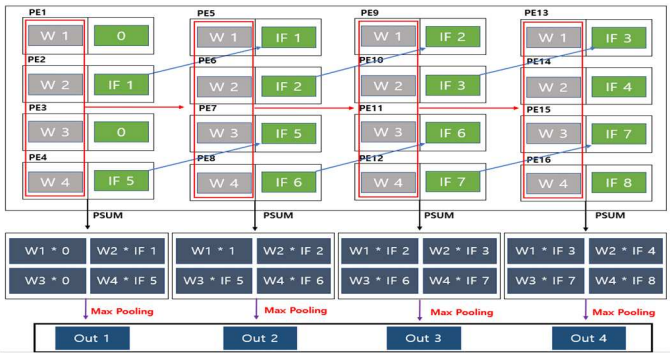

Fig. 9. CNN Reuse Operation

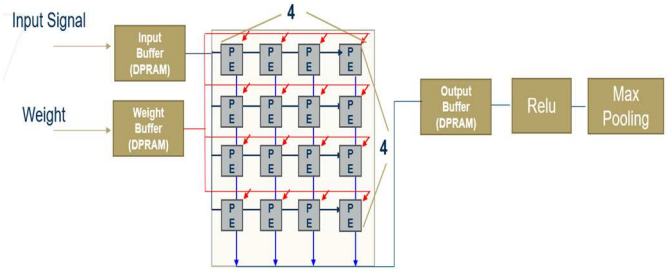

Fig. 10. CNN PE Operation

adopting an 8-bit fixed-point precision for weight representation. This strategy reduces the resource usage of multipliers and curtails memory consumption. In both blueprints, we maintain a 16-bit precision for I/O and activations.

C. *Training and Evaluation Dataset*

We train the STFT-CNN-AE using Binary Cross Entropy (BCE) loss between the original (input to the transmitter) and reconstructed (output of the receiver). AWGN with a fixed variance $\sigma_1 = (2RE_b / N_0)^{-1}$ is added to the signals during training. The model is trained on a single SNR (=6dB) but tested over a wide SNR range (0dB to 10dB). We trained 30,000 iterations with 128 batch size using Adam optimization with a learning rate was 0.001. Batch normalization was used only after the only first CNN layer.

D. *Data Quantization*

Quantization is vital for reducing model complexity in hardware accelerator design. Improved noise immunity from STFT preprocessing allows 8-bit quantization of input data and weights without compromising performance, as shown in Fig 6. Our proposed model with STFT preprocessing and quantization achieves an improved performance gain, enabling the design of efficient and resource-friendly hardware accelerators while maintaining accuracy.

E. *Accuracy Comparison*

Fig. 6 compares the BLER performance of STFT-CNN-AE, Conventional model, and MLP-AE in a 64QAM-modulated AWGN channel. BLER is determined by transmitting data blocks, using error-detecting codes to pinpoint transmission errors, and then dividing the number of error blocks by the total transmitted. STFT-CNN-AE, benefiting from STFT pre-processing and CNN architecture, achieves the same BLER as Conventional method at a 4dB lower SNR. It matches MLP-AE's performance but with lower complexity. The design's enhanced performance and reduced complexity, particularly with its lower precision BLER degradation, make STFT-CNN-AE a viable choice for efficient wireless communication with hardware constraints.

IV. STFT-CNN-AE : HARDWARE ARCHITECTURE

A. *Overall Architecture*

We integrated a CNN accelerator for the 5G autoencoder into a Zynq UltraScale+ XCZU9EG FPGA evaluation board, as illustrated in Fig. 7. This board combines a Processing System (PS) with an ARM processor and Programmable Logic (PL) linked via AXI high-performance buses. Our accelerator consists of parallel STFT and CNN modules connected to the PS. After STFT processes the data, it's moved from the PS to off-chip memory. The CNN then fetches this data, transferring it to the PL's input buffer. This parallel structure streamlines autoencoder operations. Thanks to the FPGA's adaptability, the STFT-CNN accelerator delivers both performance and flexibility, while the AXI buses ensure rapid and smooth data transfer.

A. *STFT Implementation*

For the effective execution of Fast Fourier Transform (FFT) operations, our design incorporates an array of specialized components: The Input Buffer temporarily houses the input data, thereby ensuring an uninterrupted flow into the butterfly operation module for immediate data processing. The Twiddle Buffer is dedicated to the storage of precomputed Twiddle factor values, eliminating the necessity for recalculations with each FFT operation. The Butterfly Operation Module stands as the heart of our design, where the principal FFT computations take place. Immediately following this operation, the results are momentarily held in the Interim Buffer. Within the Spectrum Value Module, the absolute values of I and Q, which denote the spectrum values, are meticulously calculated. Lastly, these computed spectrum values are stored in the Output Buffer and, in the subsequent phase, relayed to the external DDR via the advanced AXI interface. Further detailing our methodology as presented in Fig. 8, we employed the DIT Radix-4 algorithm and orchestrated an optimal balance in butterfly computations through the seamless integration of the Signal Flow Graph (SFG), segmented signal processing, and a dedicated butterfly operator. Following these computations, the power spectrum, encompassing the nuances of the I/Q output, is generated. Marrying the functionality of the Xilinx FFT block with the storage of twiddle factors as 16-bit signed fixed-points, our approach excels in both computational efficiency and providing an intricate insight into the signal's frequency landscape.

B. *CNN Implementation*

Convolutional Neural Networks (CNNs) excel in various applications, but encounter obstacles related to computational complexity, real-time latency, and resource utilization. FPGA-based acceleration offers a promising solution, offering parallel processing and customizable hardware design. This paper presents an FPGA implementation of a CNN using the row-stationary method with fixed weights. The architecture features a 4x4 Processing Element (PE) array (Fig. 9, Fig. 10) for efficient convolutional operations. In this method, we keep the weights stationary and continually feed in the input feature map. By doing so, we can efficiently compute partial sums for convolution operations in CNNs. Each psum is calculated by multiplying an element

Table 2. Comparison Resource Utilization of MLP-AE [5] and This Work

Resource	Available	MLP-AE[5]	This Work		
			STFT	CNN	Total
LUT	274,080	228,388	7,950	50,541	54,984
FF	548,160	434,560	5,863	148,832	154,695
DSP	2,520	2,210	6	60	66
BUFG	404	150	34	105	139
IO	328	266	102	31	133

Table 3. Comparison of Power Efficiency of MLP-AE and Proposed Design

Point of Comparison	MLP-AE[5]	Proposed Design	
		STFT Module	CNN Module
Maximum Frequency	75MHz	150MHz	200MHz
Latency	10.7ms	1.06us	2.44ms
Throughput	37.38MPOS	3,750MOPS	133MOPS
Dynamic Power	3.46W	650mW	1.758W
Static Power	2.85W	480mW	0.84W
Total Power	6.31W	1.13W	2.598W

of the input feature map with a corresponding weight and then adding it to the previous psum. After these computations, we apply max pooling, which selects the maximum value from each sub-region of the input feature map, effectively downsizing it while preserving important information. This strategy allows us to take advantage of FPGA's parallel processing capabilities and achieve efficient computation for CNNs. It eliminates weight transfer during computation cycles, reduces memory access latency, and employs sequential input feeding. Concurrent processing within the PE array maximizes hardware resource utilization and allows simultaneous computations. The CNN block operates on the power spectrum from the STFT module, with layer weights stored in off-chip memory. To optimize hardware utilization, different layers share the same accelerator. The dedicated PE array performs activation computations in a single cycle; however, the number of cycles required varies depending on the layer complexity and total number of layers.

1) First Convolution Layer: The first convolution layer's input data size is 8 x 25. It consists of 60 kernel filters, each having a kernel size of (2 x 2) and a stride of 1. With a 16 PE array and a kernel size of 2 x 2, the layer calculates four output partial sum (psum) data per cycle. Input rows are vertically summed to generate partial sums for one row, reusing weights and storing partial sums in the output buffer. After max-pooling, the final output data is obtained. Zero padding is applied to maintain the input size post-convolution. The first convolution layer requires a total of 12,000 cycles (number of kernels x number of cycles = 60 x 200). Given a cycle latency of 5 ns, the total latency for the first convolution layer is 60 μs.

2) Second Convolution Layer: The input of the second convolution layer is the output of the first convolution layer, which has the same size. This layer consists of 30 kernel filters, each with a kernel size of (2 x 2) and a stride of 1. Using a 16 PE array of size 4 x 4, the output data is computed over a total of 360,000 cycles (number of kernels x number of cycles = 60 x 30 x 200). Considering that the latency of one cycle is five ns, the total latency of the second convolution layer is 1.8 ms.

3) Third Convolution Layer: Using the same approach as described above, the input of the third convolution layer is the output of the second convolution layer, maintaining the same size. This layer is comprised of 15 kernel filters, each with a (2 x 2) kernel size and a stride of 1. By utilizing a 16 PE array with dimensions of 4 x 4, the output data is computed over a total of 90,000 cycles (30 x 15 x 200). With a latency of 5 ns per cycle, the overall latency for the third convolution layer is 440 us.

4) Fourth Convolution Layer: Using the same approach as described above, this layer is comprised of 5 kernel filters, and the output data is computed over a total of 15,000 cycles (15 x 5 x 200). With a latency of 5 ns per cycle, the overall latency for the fourth convolution layer is 75 us.

5) Fully Connected Layer: The fully connected layer has 200 nodes. By utilizing the existing PEs, it requires 13,000 cycles (calculated as 200 x 5 x 200/16) to complete the computations. The latency for the fully connected layers is 65 us.

C. FPGA Implementation of baseline MLP-AE

In our study, we opted for a fully parallel architecture to implement the MLP-based Autoencoder (MLP-AE) on FPGA platforms. This design connects parallel processing elements to every node within individual layers, enhancing throughput by allowing simultaneous computation and minimizing latency by reducing data dependencies between computations—an approach that contrasts with the conventional row stationary dataflow commonly preferred for CNN accelerators [8] due to its adaptability to convolutional tasks.

For the FPGA adaptation of the MLP-AE, weights, and activation functions have been meticulously tailored for each layer to ensure optimal parallel computation. This Autoencoder configuration comprises five layers: the first four encompassing 512 neurons each and a fifth layer housing 200 neurons dedicated solely to signal reconstruction. It's important to note that latency in a CNN layer depends largely on factors such as its overall architectural design—like how many layers it has or how densely they are connected—and specific settings like kernel size or stride length used in convolution operations. In this research, we contrasted the FPGA execution of MLP-AE against our proposed STFT-CNN-AE model, where we adopted a parallel data-flow blueprint—assigning one processing element (PE) per neuron across all layers. The projected latency for this MLP-AE setup was estimated using simulations based on hardware specifications and model parameters—it came out around 10.7 milliseconds (ms).

D. Comparison of MLP-AE and STFT-CNN-AE

We juxtapose the FPGA execution of the advanced STFT-CNN-AE system with the previously established FPGA-based MLP-AE [5].

Table 1 offers a side-by-side examination of the two architectures. Notably, the STFT-CNN-AE necessitates fewer parameters and calculations, which subsequently reduces the demand on FPGA resources.

Table 2 provides a closer look into MLP-AE's and STFT-CNN-AE's resource consumption. The diminished computational requirements of STFT-CNN-AE translate to less consumption of Look-Up Tables (LUTs) and Digital Signal Processing (DSP) components.

Furthermore, Table 3 sheds light on the distinct differences in power dissipation, throughput, and overall performance of the two models. Impressively, the STFT-CNN-AE boasts a throughput of 133 MOPS, overshadowing the MLP-AE's throughput of 37.38 MOPS. By opting for decreased weight precision and more compact Processing Elements (PEs), the STFT-CNN-AE, furnished with 8-bit weights, reaches a higher maximum frequency in contrast to the MLP-AE's 16-bit weights. This surge in frequency, in tandem with reduced computational demands, culminates in a substantially decreased processing latency for STFT-CNN-AE, which is gauged by multiplying the cycle number with the cycle time. Regarding power consumption, the STFT-CNN-AE and MLP-AE record total power consumptions of 2.598W and 6.31W, respectively. Intriguingly, even when factoring in the power overhead of STFT, our proposed architecture achieves a commendable 30% reduction in dynamic power and a substantial 54% decrement in static power relative to the MLP-AE.

VI. Conclusion

This study presents the STFT-CNN-based autoencoder tailored for 5G communication systems. The paper also presents an FPGA implementation of STFT-CNN-AE design and a baseline MLP-AE. The two designs' resource utilization, power dissipation, and performance are compared, showing the benefits of STFT-CNN-AE over MLP-AE. The STFT-CNN-AE demonstrates reduced resource requirements due to its lower parameter count and computational demands, resulting in lower utilization of Look-Up Tables (LUTs) and Digital Signal Processing (DSP) resources, making it a more efficient choice for implementations. Additionally, STFT-CNN-AE achieves a higher maximum frequency owing to reduced weight precision and smaller Processing Elements (PEs), leading to significantly lower processing latency than MLP-AE. Regarding power consumption, STFT-CNN-AE exhibits superior power efficiency with a 59% reduction compared to MLP-AE. These power savings result from the reduced precision computation and decreased resource requirements of the STFT-CNN-AE architecture. The STFT-CNN-AE's FPGA implementation offers improved resource utilization, lower power consumption, and reduced processing latency, making it a compelling choice for autoencoder applications in FPGA-based wireless communication systems.

References

[1] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," IEEE Transactions on Cognitive Communications and Networking, vol. 3, no. 4, pp. 563–575, 2017.

[2] T. J. O'Shea, T. Erpek, and T. C. Clancy, "Deep learning based mimo communications," arXiv preprint arXiv:1707.07980, 2017.

[3] C. Sebastian, et al. "Trainable communication systems: Concepts and prototype." IEEE Transactions on Communications 68.9 (2020): 5489-5503.

[4] D. J. Ji, J. Park, and D.-H. Cho, "ConvAE: A New Channel Autoencoder Based on Convolutional Layers and Residual Connections," IEEE Commun. Lett., vol. PP, no. c, pp. 1–1, 2019.

[5] M. Kim, W. Lee, Jungmin Yoon, Ohyun Jo, "Building Encoder and Decoder with Deep Neural Networks: On the Way to Reality" arXiv:1808.02401 [cs.IT]

[6] S. Cammerer, F. Ait Aoudia, S. Dörner, M. Stark, J. Hoydis and S. ten Brink, "Trainable Communication Systems: Concepts and Prototype," in IEEE Transactions on Communications, vol. 68, no. 9, pp. 5489-5503, Sept. 2020, doi: 10.1109/TCOMM.2020.3002915.

[7] N. Wu, X. Wang, B. Lin, and K. Zhang, "A CNN-Based End-to-End Learning Framework Toward Intelligent Communication Systems," IEEE Access, vol. 7, pp. 110197–110204, 2019.

[8] Y. -H. Chen, J. Emer and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 367-379, doi: 10.1109/ISCA.2016.40.

[9] D. Kim, T. Na, S. Yalamanchili and S. Mukhopadhyay, "DeepTrain: A Programmable Embedded Platform for Training Deep Neural Networks," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2360-2370, Nov. 2018, doi: 10.1109/TCAD.2018.2858358.

[10] Z. Cong, et al. "Channel autoencoder for wireless communication: State of the art, challenges, and trends." IEEE Communications Magazine 59.5 (2021): 136-142.

[11] W. Dehao, Maziar Nekovee, and Yue Wang. "Deep learning-based autoencoder for m-user wireless interference channel physical layer design." IEEE Access 8 (2020): 174679-174691.

[12] M. Nadya A., and Joseph R. Cavallaro. "Design and Implementation of Autoencoder-LSTM Accelerator for Edge Outlier Detection." 2021 IEEE Workshop on Signal Processing Systems (SiPS). IEEE, 2021.

[13] H. Zhaohui, et al. "Autoencoder with fitting network for Terahertz wireless communications: A deep learning approach." China Communications 19.3 (2022): 172-180.