Invited Paper: Accelerating Next-G Wireless Communications with FPGA-based AI Accelerators

Chunxiao Lin
ECE Department
Virginia Tech
Blacksburg, VA, US
chunxiaol@vt.edu

Muhammad Farhan Azmine
ECE Department
Virginia Tech
Blacksburg, VA, USA
Muhammadfarhan@vt.edu

Yang (Cindy) Yi
ECE Department
Virginia Tech
Blacksburg, VA, USA
Yangyi8@vt.edu

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323918

***Abstract*—5G and beyond 5G wireless communication has revolutionized our daily lives. However, the increased bandwidth and data rate transfer in 5G present challenges, particularly in the domain of data receiving and recovery tasks. Orthogonal frequency-division multiplexing (OFDM) Symbol detection plays a pivotal role in ensuring efficient and error-free transmission in next-G communications. To ensure optimal performance and address potential bottlenecks and errors, propose a Field-Programmable Gate Array (FPGA)-based AI accelerator to accelerate symbol detection, which holds immense potential for enhancing signal recovery in MIMO systems and unlocking the full efficiency of 5G technology. To be more specific, we employ a hardware-verified Echo State Network (ESN) as the symbol detection method in the multiple-input and multiple-output (MIMO)-OFDM system. The ESN, a hardware-efficient Recurrent Neural Network, features a fixed reservoir network structure and fewer trainable parameters in the output layer. To validate our approach in real time, we construct a Software-defined Radio (SDR) platform. This platform allows us to collect datasets in real-world scenarios using antennas. Our experiment showcases a promising average bit-error rate (BER) of .04 in the performance of our FPGA-based design under realistic conditions. We were able to achieve 3.3 times the throughput with an increase of only 21.4% LUT and 33% FF in resource utilization for maximal processing speed design which is relatively lower in comparison with the ESN implementation for SISO systems. [1] In addition, we reduced the BRAM memory and DSP IP usage by 50% and 33.3% respectively.**

***Index Terms*—RNN, AI accelerator, FPGA, Next-G communication, 5G, MIMO-OFDM, echo state network**

I. Introduction

With the way wireless connectivity and Internet of Things (IoT) devices have expanded over the last few years, 4G networks are gradually reaching their limits. This accelerates the need for faster and more efficient wireless technology that has the bandwidth to accommodate larger volumes of traffic.

This brings us to 5G networks, which is the proposed upgrade to present-day wireless communication, which is slowly being rolled out through all devices as a standard for wireless communications. MIMO-OFDM **(Multiple Input Multiple Output - Orthogonal Frequency Division Multiplexing)** is one of the core wireless communication interfaces in 5G networks that exploit both spatial and temporal dimensions for symbol interference. The improvement in the symbol detection process can play a crucial role in the Next-G communication technology. The conventional methods for symbol detection suffer from several issues including unreliability due to reliance on channel state information (CSI) which can change dynamically due to environmental effects. [1] The Machine-learning-based methods can circumvent this issue since they do not require CSI data during real-time communication operations.

Machine Learning (ML) is one of the most popular methods of building models based on data and has been considered a substitute for the traditional model-based approaches in many aspects of wireless communication systems - from channel coding and transceiver design to modulation and signal classification. Also, ML-based methods are supposedly better candidates than the traditional ones in dealing with systems with exponentially increasing complexity, such as MIMO communication [2]–[4]. Unfortunately, data-driven ML methods always require large amounts of input data **i.e.** ***Data Deficit*** for achieving a satisfactory accuracy level. Furthermore, in massive MIMO, the general Neural Network models require a high amount of trainable parameters to achieve promising accuracy which in return makes the model more complex to be implemented inside the hardware. Also, more neuron parameters can result in over-fitting with slower convergence and exhibit poor generalization performance. [5].

In this paper, we address these challenges in designing an FPGA **(Field-Programmable Gate Array)** ESN **(Echo State Network)** hardware accelerator, which is combined with the SDR **(Software-defined Radio)** transceiver and tested in real-world scenarios. Specifically, we have the following contributions:

1) We demonstrated the feasibility of utilizing an FPGA-based AI accelerator for the symbol detection task in the MIMO-OFDM system.
2) We constructed an efficient ESN model with optimized hyperparameter tuning for the high accuracy symbol detection task of our selected application.
3) We proposed an FPGA-based architecture that explores full capacity exploitation of DSP (Digital Signal Processing) units for implementing the designed AI model.
4) Our accelerator was capable of achieving promising accuracy with low resource utilization and power consumption in our FPGA/SDR real-time platform.

II. MIMO-OFDM Symbol Detection

A typical MIMO-OFDM communication system with N_t number of transmitters and N_r number of receivers as can be observed in Figure 1. MIMO facilitates simultaneous transmission of multiple signals, while OFDM divides these signals into subchannels for parallel transmission.

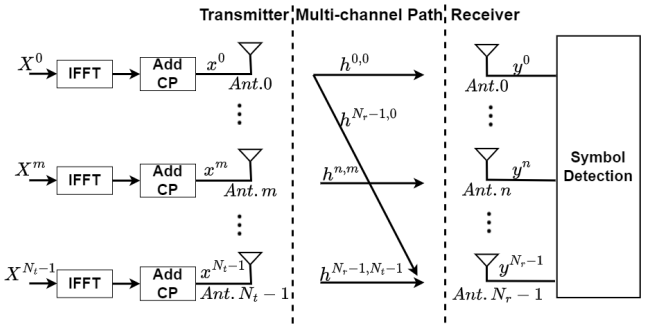

Fig. 1. Generic Network view of MIMO-OFDM communication

In the OFDM symbol Frame, reference symbols are embedded for symbol detection. The first N_{TS} symbols of the frame are designated as training sequences, which are already known by the receiver before transmission. Consequently, the training sequence received at the receiver antenna can be expressed with the following equation for the case of k^{th} sub-carrier:

$$\mathbf{Y}_k^{TS} = H_k X_k^{TS} + G, \tag{1}$$

A figure of the OFDM frame structure consisting of the training sequence and data structure is provided below.

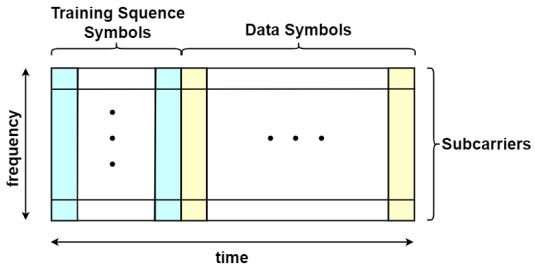

Fig. 2. Frame structure of the OFDM Signal

Here X_k^{TS} is the transmitted training sequence with a matrix dimension of $X_k^{TS} \in \mathbb{R}^{N_t \times NT_S}$, and $Y_k^{TS} \in \mathbb{R}^{N_r \times NT_S}$ are the received TS symbols. G is the Channel Noise that has a probability distribution similar to the Gaussian distribution with a variance of σ^2.In the equation, H_k represents the channel response estimation for the signal transmitted at k^{th} sub-carrier.

In conventional symbol detection methods like the Linear Minimum Mean Square Method (LMMSE), the wireless channel model is estimated and then used for signal recovery. The details are shown as follows.

1) In the first step, with the help of the predetermined-transmitted and the channel propagated-received training sequence, the estimated channel response for the k^{th} sub-carrier can be extracted using the following relationship :

$$\hat{H}_k = \frac{Y_k^{TS} X_k^{TS*}}{X_k^{TS} X_k^{TS*} + \sigma^2 I}, \tag{2}$$

2) After extracting the channel response $\hat{H}_k$, the sent data of $i_t h$ OFDM symbol for sub-carrier k can be recovered from the received data $Y_k(i)$ using the following equation :

$$\hat{X}_k(i) = \frac{\hat{H}_k * Y_k(i)}{\hat{H}_k * \hat{H}_k + \sigma^2 I}, \tag{3}$$

However, the LMMSE still has some limitations since it needs correct channel estimation response value as well as the foreknown data of the noise variance and channel statistics. The channel estimation can only be done after getting the received training sequence at the receiver end. This makes the method limited for the cases of changing environmental conditions and also for a low Signal to Noise ratio (SNR) in the channel path. The performance loss as a result of the inefficacy of channel response computation will be shown in the results as real-world experiment cases.

Besides LMMSE, there are several other methods available for symbol detection which are inferior because of their implementation complexity. Some of the methods are written as follows :

- **The Maximum Likelihood Method:** This method is able to achieve accurate results with minimal error for the symbol detection task. However, the model is not used in practical applications of its exponential complexity.
- **Sphere Decoding (SD):** The SD method is efficient in maximum-likelihood detection, but the complexity increases with a decrease in SNR ratio and thus makes the lower complexity model of LMMSE more preferable.

The ESN-based method for symbol detection can circumvent the limitations of the above methods, especially the adaptability issue of channel response estimation and the need for prior noise variance data of the channel path for the LMMSE model. ESN-based symbol detection has also been proven to be energy-efficient for a total transmitter-receiver power system than the LMMSE model resulting in lower energy consumption per information bit [6].

III. Related Works

Researchers have investigated the FPGA in Next-G communication applications, such as [7], [8], implementing FPGA accelerators for soft-output data detection in large-scale MIMO-based LTE systems. However, limited work addresses FPGA accelerators for symbol detection in MIMO-OFDM systems, which is a core component for data recovery in reception.

The integration of AI to cellular connectivity is also under exploration due to its high adaptability to changing environments and capability to map the complex characteristics of data transmission to higher dimensions [9]. Theoretical

approaches have been explored for using ESN-based AI symbol detection since they can be efficient in such types of operations. [10]

In our previous work [1], [11], FPGA accelerators with different architectures have been explored for the SISO-OFDM symbol detection task, underscoring the superiority of ML-based symbol detection methods over traditional approaches. In this study, our focus is on designing an FPGA accelerator for ESN-based symbol detection within a real-time MIMO-OFDM system, facing challenges from a more complicated wireless channel and high transmission speed requirements. Such design specifications demand high processing throughput capacity in the hardware along with mitigating the over-fitting issue of ML networks due to larger data input size. Moreover, most FPGA-related research on this topic did not investigate the efficiency of the IPs to its full extent [12], [13]. Here, we also exploit the DSP IPs of the FPGA board in MIMO-OFDM for the first time.

IV. AI Model for Next-G Communication

The architecture of the RNN-based ESN model is shown in figure 3. The time-serial input data is first processed by a pre-designed weight matrix, denoted as W_{in}. The transformed data is then mapped into an RNN-based reservoir, where it is encoded as a vector of high-dimensional patterns stored in the reservoir neurons. Subsequently, a pattern analysis stage is carried out, utilizing a trainable weight matrix, denoted as W_{out}, to generate the desired outputs.

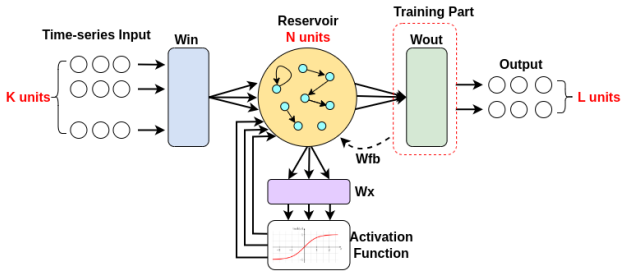

Fig. 3. The Echo State Network with an RNN-based reservoir. There are K input units, N reservoir units, and L output units. For dimensions of the weight matrices, $W_{in} \in R^{N\times K}$, $W_x \in R^{N\times N}$, $W_{out} \in R^{L\times N}$.

As a special type of recurrent neural network, ESN incorporates a fixed reservoir as its core component. The reservoir updating is processed with the following equation 4:

$$\mathbf{r}_{t+1} = f_{nonlin}(W_{in}\mathbf{x}_{t+1} + W_x\mathbf{r}_t + W_{fb}\mathbf{y}_t), \quad (4)$$

Here x_t and y_t are the current input and output of the network, while the x_t and x_{t+1} are the reservoir states of the current step and the following step. W_{in}, W_x, and W_{fb} are the weight matrices representing the fixed connections between different layers.

The readout of the ESN model is then generated with another calculation of the sum of products represented by equation 5 in matrix format.

$$\mathbf{Y}_{t+1} = f_{out}(W^{out}X). \quad (5)$$

Here X is a concatenated matrix of the $\{x_{t+1}; r_t; y_t\}$. The output activation function f_{out}, if taken as a linear function, can perform with minimal precision. In this way, ESN can become a better candidate in comparison to Back-propagation Decor-relation (BPDC) in RNNs, a similar learning rule derived from ESN.

A. Training Model and Prediction Methods

The only trainable parameters of the ESN are the output weights W_{out} which are trained through back-propagation based on the loss function of MSE (Mean Square Error) between the predicted outputs and the training labels. If the training label matrix is declared as $Y_{desired}$ and W^{out} is the training parameters matrix of the ESN network, then it can be expressed as,

$$\mathbf{W}^{out} = Y_{desired}X^T(XX^T + \beta I)^{-1} \quad (6)$$

Here the β is the ridge regression i.e. regularization factor which is dependent on the learning rate of the training optimizer loss function.

After training for a sufficient number of iterations, the W^{out} can be extracted and used for inference for prediction of symbol detection operation using the equations of (4) and (5).

V. Proposed FPGA Accelerator Architecture

To maximize the processing speed of our design, we decided to fully exploit the DSP48E1 IP blocks in our design in a methodical approach, which are dedicated hardware units designed for accelerating the performance speed and accuracy of arithmetic operations related to digital signal processing (e.g. multiplication, addition, subtraction, and accumulation). These IPs are provided by the vendors in the Xilinx Virtex-7 series FPGA board that we choose. As shown in Figure 4, it consists of a power-saving pre-adder, a 25 x 18 two's complement multiplier, a 48-bit accumulator, a 3-input **SIMD** (i.e. Single-Instruction-Multiple-Data) arithmetic logic unit **(ALU)** and a pattern detector.

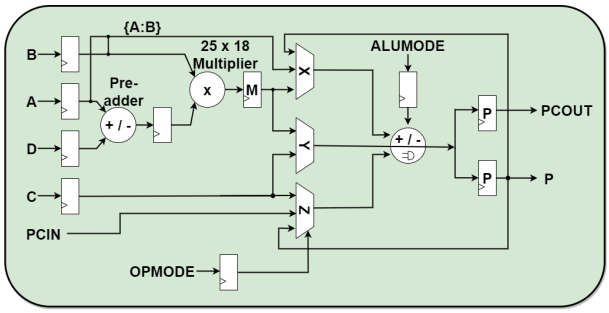

Fig. 4. Overview of key-characteristics of DSP48E1 unit

There are 4 input registers as **A, B, C, D**, and the final output of the block is stored in the **P** register as seen from the figure. There is a provision for concatenating the **{A: B}**

registers up to 48-bit which can be directly sent as input to the SIMD-ALU. Also, the slice has a high-speed inter-connection input-output port of **PCIN/PCOUT** which can be exploited for managing pipeline stages and facilitating parallel execution of operations using multiple DSPs. These ports cannot be used anywhere except among DSP IPs.

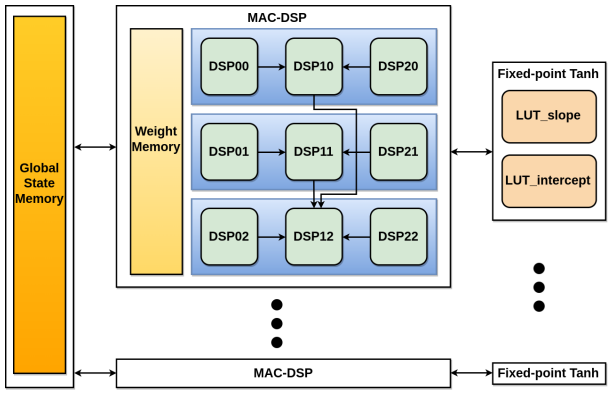

Fig. 5. Reservoir neuron architecture with the MAC-DSP unit

Back to the reservoir updating operation in the ESN system, equation 4 can be rewritten into equation 7 with concatenations on the weight matrices and data vectors separately:

$$\mathbf{r}_{t+1} = f_{nonlin}(W_{ex}\mathbf{r}_{ex}), \tag{7}$$

where we consider an ESN reservoir without any feedback connections and W_{fb} from the output layer to the reservoir layer. W_{ex} is then a concatenated weight matrix of $\{W_x; W_{in}\}$ and $\mathbf{r}_{ex}$ is the concatenated data vector of $\{r_{t+1};\ x_{t+1}\}$.

The calculation of Equation 7, involves a matrix-vector multiplication operation and a non-linear function approximation. Our architecture successfully meets the demand of both ***(K+N)(i.e 48)*** number of multipliers and ***(K+N-1)(i.e 47)*** number of adders along with the non-linear function operation with only 9 DSPs for each neuron as shown in Figure 5. It significantly decreases the use of custom logic CLBs which leads to a power-efficient and high-performance processing architecture since the dedicated IP hardware blocks have optimized designs along with specialized processing pipelines having improved latency which exceeds the performance of components implemented with regular CLBs. The input and reservoir weights related to the specific neuron are loaded into the local weight memory of that neuron whereas the extended state is saved in the global state memory which can be accessed by all the neurons according to Figure 5.

The 9 DSPs are configured sequentially in different pipeline stages for performing the required operations by each neuron of the reservoir. The first 3 pipeline stages are configured in the DSPs for performing the matrix-vector multiplication needed by Equation 7. These stages are explained below:

- Stage I: The first stage uses all the DSPs individually to concurrently import, multiply, and accumulate 9 groups of weights and states loaded from the local weight memory and extended global state memory. A brief view of the active path used in Stage I of the DSP is shown in Figure 6(a) which is also designated as **Type_A**. In the first clock cycle, the inputs of **A** and **B** are multiplied by the dedicated multiplier inside the DSP block and stored in the **M** register. In the second clock cycle, the current value of **M** register is accumulated with the previously stored value and stored inside the **P** register. So this step takes 2 clock cycles to process its **MAC**(Multiply and Accumulate) operation. The function of each DSP in Stage I can be represented by the following equation :

$$\mathbf{P} = \mathbf{P} \pm (\mathbf{B} \times (\mathbf{A})) \tag{8}$$

- Stage II: In this stage, each 3 DSP blocks are connected as a group using the high-speed PCIN/PCOUT ports. As shown in Figure 6 (b), the upper and lower **Type_A** DSP inputs are shorted to zero so that the values inside the P registers of those DSPs can be retained.
  The **Type_B** DSP (Active path. Figure 6.c)in this stage is configured to perform the following equation:

$$\mathbf{P} = \mathbf{PCIN} + \mathbf{C} + \mathbf{P}, \tag{9}$$

  The **SoP** expression of the upper Type_A DSP (P_{00}) takes one clock cycle to be transferred into the **C** register of Type_B DSP. In the next clock cycle, the SoP (P_{20}) from the lower Type_A DSP is sent to the 3-input SIMD-ALU along with the other two SoPs $(P_{00}\&P_{10})$ stored already in P & C registers in the Type_B DSP block for compressing them into a single expression which is finally stored in the P register of the Type_B DSP : $P_{10} = P_{00} + P_{10} + P_{20}$
- Stage III: In the 3rd stage, the 3 SoPs (P_{10}, P_{11}, P_{12}) in Figure 6(d) produced from the three pipelined DSPs of Stage II are finally combined to one SoP by a single Type_C DSP from Figure 6(d) block which already has SoP P_{12} saved in its P register as a consequence of its operation result from Stage II. The rest of the two SoPs (P_{10}, P_{11}) are sent as input to the mentioned Type_C DSP block. According to the Xilinx DSP48E1 user manual, the **A** & **B** registers have bit-width limitations of 30 and 18 bits respectively whereas the **C** & **P** registers both can accommodate 48-bit width data. The provision of concatenation of both **{A:B}** registers is used to take one of the SoPs P_{10}. The P_{11} SoP is sent through the register **C** into the final DSP block which takes one clock cycle. The addition is performed in the second clock cycle by the 3-input SIMD-ALU which formulates the 10 given below :

$$\mathbf{P} = \{\mathbf{A} : \mathbf{B}\} + \mathbf{C} + \mathbf{P}, \tag{10}$$

  The combined result of the SoPs is stored in the output of Type_C register and can be written as $P_{12} = P_{10} + P_{11} + P_{12}$.
  All the results can be processed in $[(K + N)/9]$ +1 ***(for Stage I)*** +2 ***(for Stage II)*** + 2 ***(for Stage III)*** in

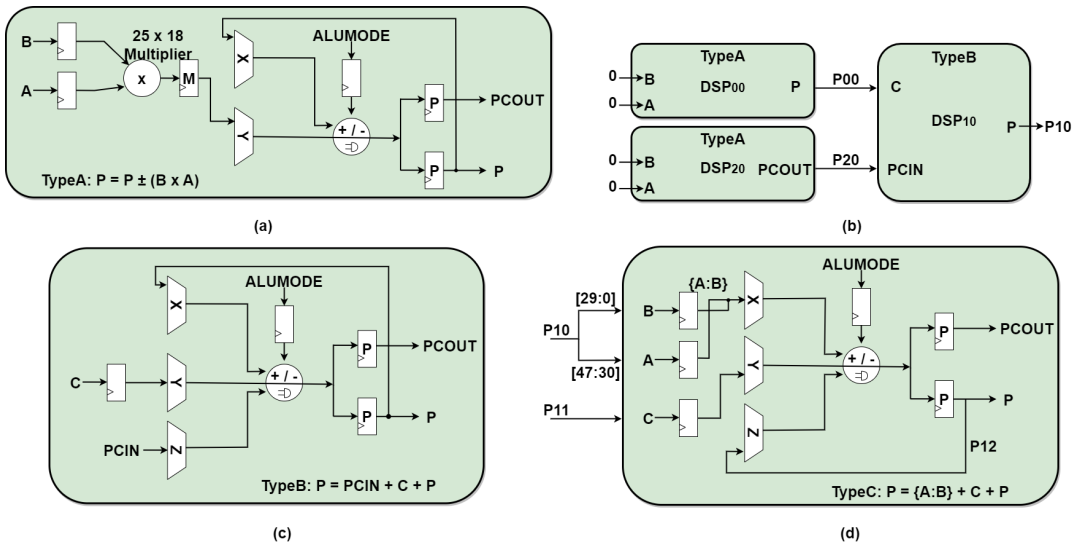

Fig. 6. (a) Active path of Type_A DSP, used in MAC operation of Stage I (b) Overview of DSP connections in Stage II; the output of Stage I Type_A DSPs are sent to Type_B DSP for Compression of Stage II (c) Active path of Type_B DSP used in Stage II (d) Active path of Type_C DSP used in Stage III

the above steps which were required for Matrix-vector multiplication in the ESN reservoir.

The final SoP is sent to the DSP_{22} in Figure 5 which performs the process of non-linear function operation.

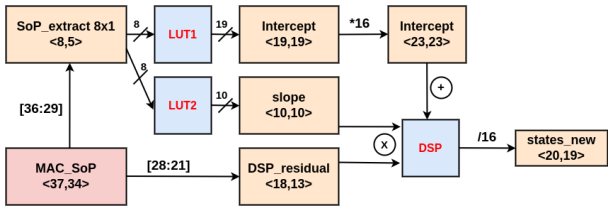

Fig. 7. Fixed Point Non-linear tanh operation

For the non-linear activation function, we implemented a fixed-point hyperbolic tangent ($tanh$) function, which is also shown in Figure 7. The $tanh$ function is simplified by its first-order piece-wise linear approximation in the format below:

$$\tanh(x) \approx \begin{cases} -1 & \text{for } x \leq -a \\ mx + b & \text{for } -a < x < a \\ 1 & \text{for } x \geq a \end{cases}, \qquad (11)$$

Here, a is a positive constant that determines the range of linear approximation. m and b are the slope and intercept of the linear approximation decided by the value of a. In our work, two look-up tables (LUTs) are implemented to store the slope and intercept using Block RAMs (BRAMs). For generating constants, a simplified method of Bajger and Omondi is implied. [14] The final SoP of the MAC operation will be used to index the two LUTs. Then another DSP block in the MAC-DSP unit is assigned to calculate the $tanh$ linear approximation with the slope and intercept generated from LUTs.

To optimize the utilization of our design without getting a much worse accuracy, we implemented our work in a fixed-point system. Considering the input bit-width of the DSP multiplier, we limit the W_{out} to the bit-width of 16. However, we still need to make sure the precision of W_{out} is good enough to get high-accuracy performance. By trading off between the precision of $tanh$ function and W_{out}, we find that good accuracy can be achieved by setting the LUT precision to 2^{-6} and the W_{out} precision to 2^{-5}. To achieve this, we carefully selected the values for the parameter and listed them in Table I.

TABLE I
PARAMETER SETTING OF THE PROPOSED HARDWARE ESN DESIGN

Parameter	*Value*
Number of inputs	40
Number of reservoir neurons	8
Number of outputs	4
Range of inputs	(-0.1,0.1)
Range of output weights	(-700,700)
a (in Tanh approximation)	8
Depth of Tanh LUT	2^8

In our design, we set the number of inputs and outputs to 40 and 4, respectively, according to the structure of the MIMO system. To determine the reservoir size, we evaluate the symbol detection task performance using different numbers of neurons. The results show that an ESN with a reservoir size of 8 is the best option, offering the second-best BER **(Bit-error-rate)** performance and low resource utilization in subsequent hardware design. Further details on the reservoir size selection will be provided in the experiment results section.

Then for the two Tanh LUTs, we set the range of linear approximation a to 8, which keeps the error out of the range within 10^{-6}. To achieve the precision 2^{-6}, the depth of the LUTs is designed to be 2^8, which guarantees a relatively small

resource utilization in the meantime.

Finally, we normalize the range of inputs to (-0.1, 0.1) to adjust the output weights W_{out} into a range of (-700,700). In this way, the 16 bits of output weights can be divided into a sign bit, an integer part of 10 bits, and a fractional part with 5 bits.

VI. Experimental Results

A. Experiment Setup

To validate our FPGA-based AI accelerator for MIMO-OFDM symbol detection, we set up a real-time platform of FPGA board and SDR simulation software, along with the Universal Software Radio Peripheral (USRP) devices. In the setup shown in Figure 8, a PC-FPGA configuration with four USRPs was implemented. The USRPs are used as RF transceiver antennas. The Wi-Fi radio system is implemented in the GNU Radio SDR framework which generates the transmitting signal data and sends it directly to the transmitter USRP. The transmitter then sends the signal through the wireless channel to the receiver USRP where the received signal is directly sent to the FPGA. After that, our FPGA-based ESN is used for OFDM symbol detection and recovery of the transmitted signal. The recovered signal is finally sent back to the host PC for comparing it with the transmitted symbols and investigating the bit error rates. In the radio system, conventional MMSE-based symbol detection is also implemented to be compared with our FPGA-ESN-based symbol detection method.

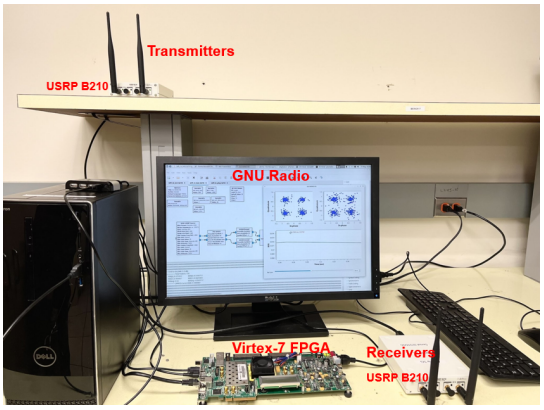

Fig. 8. The real-time FPGA/SDR jointed platform for 2x2 MIMO-OFDM symbol detection

For this experiment, the Xilinx Virtex-7 VC707 FPGA board was chosen, which provides an abundant amount of DSP slices (i.e. 2800) and low latency-Faster clock capacity for high-speed wifi data transmission. Also, The Virtex-7 FPGA board we chose does not rely on an internal CPU, avoiding the limitations of other FPGA series that face slow data transfer [15], and enabling optimized design with higher throughput and lower resource utilization.

To validate the performance of our design in real-world scenarios, we have the tests carried out in our RF lab, where we have many experiments in wireless communication [16]. There are severe multi-path scattering effects for the reception that will make the environment more realistic. For the diversity of the testing scenarios, various metal shelves and toolbox cases are used to block the transmitter/receiver line-of-sight (LoS). There are 5 different test scenarios we designed for the whole experiment. The details of them are shown in the table II. Each test case runs for 3 trials, with the same antenna orientation and near identical test setup. Moreover, the antenna orientation and Tx/Rx placements are changed from test to test, with different receiver RF front-end gains.

TABLE II
The Setup of Different Scenarios

Scenario	*Description*
Scenario 1	Tx-Rx 10m apart (long distance), clear line-of-sight (LoS) signal path.
Scenario 2	Tx-Rx 5m apart with non-line-of-sight signal path (NLoS), the transmitter and receiver are partially blocked.
Scenario 3	Tx-Rx placed further apart (NLoS), the transmitter and receiver are completely blocked.
Scenario 4	Tx-Rx placed further apart (NLoS) and partially blocked.
Scenario 5	Tx-Rx placed furthest apart, the transmitter and receiver are completely blocked.

B. Results and Analysis

In conventional neural network design, the number of layers and neurons significantly influences the accuracy of the proposed model. We conducted an experiment on the impact of reservoir size on final BER results in fixed point simulation. In each of the five scenarios, different numbers of reservoir neurons are applied to the ESN model. We then calculate the average BER across all scenarios to establish the relationship between the average BER and reservoir size, as depicted in Figure 9.

The chart reveals that ESN with 16 reservoir neurons achieves the lowest BER of 2.2606, while ESN with 8 reservoir neurons attains the second lowest BER of 2.2608%. Considering later hardware design, ESN with a reservoir size of 8 offers substantially lower resource utilization, making it the best option for our design.

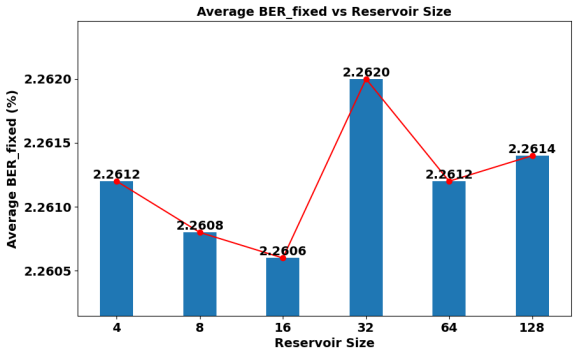

Fig. 9. BER performance for different reservoir size in fixed point simulation

In Figure 10, we display the BER performance of the conventional LMMSE method, the software ESN-based method, and the hardware ESN-based method in five different scenarios listed above. As discussed in Section II, the other two symbol detection methods(i.e. maximum likelihood optimal method and sphere decoding method) are not included in the baseline list for comparison because of their high-time complexity, which hinders practical implementation in wireless systems.

The five scenarios are designed to give a complete performance evaluation under different real-world situations. Scenario 1 is a line-of-sight signal path in a long distance, and scenario 2 is a non-line-of-sight channel in a short distance. Both of them show very good reception with extremely low BER ($< 1\%$). The BER difference between them is not significant since the multi-path effect on the transmitted signal is small in scenario 2. Scenario 3 is a non-line-of-sight path that is fully blocked by obstacles. It induces a rapid multi-path fading which results in a degradation of the BER performance. Scenario 4 is still a non-line-of-sight path with a long distance. Compared to scenario 3, the signal, in this case, is partially blocked to reduce the multi-path effect. Then it achieves a slightly better BER, which is under our expectations. The last scenario is a non-line-of-sight channel with the longest distance, and transceivers are completely blocked as well. BERs of all the methods are highly increased due to the severe multi-path fading effects.

The hw_ESN is based on the fixed-point design of the sw_ESN, which is implemented in Python with floating-point values. We can learn from Figure 10 that the difference of BER between hw_ESN and sw_ESN in all scenarios is within 0.03%. This proves the high precision of our FPGA-based ESN design. Moreover, both ESN-based methods perform a better BER performance in all the scenarios, and the advantage of ESN-based methods is more significant in the cases with worse channel situations.

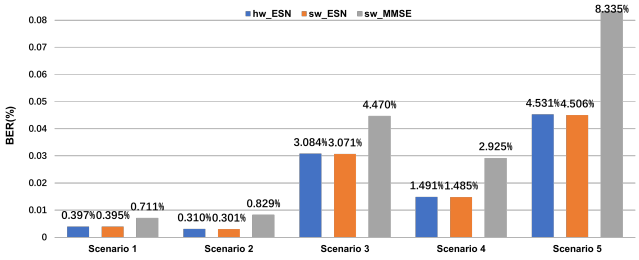

Fig. 10. The BER performance of different symbol detection methods in 2x2 MIMO-OFDM

C. FPGA Synthesis Results

The ESN-FPGA hardware for MIMO-OFDM proved to be power efficient as per the standard of the VC-707 board with a central clock speed of 125 MHz. The overall design with maximum throughput showed 0.256 W of dynamic power consumption and 0.262 W of static power consumption in the Vivado power report tool where total on-chip power consumption is about 0.754 W. The design also demonstrates a junction temperature of 25.9 degrees (Celsius) which is well within the limit of having the capability to stand up to a maximum temperature of 85°C, according to the Vivado report.

The architecture was synthesized with no timing violation since there was zero negative slack for all types (i.e. Setup, Hold & Pulse Width time) of timing requirements.

The utilization of different block units for the Vivado VC-707 board is exhibited in the following table for our ESN-AI accelerator architecture :

TABLE III
RESOURCE UTILIZATION SUMMARY IN ESN-ACCELERATOR

Optimization type	Max Processing Speed
LUT	13314 (4.9%)
FF	10750 (1.77%)
BRAM	6 (.58%)
DSP	108 (3.86%)

Our MIMO-ESN accelerator implementation on VC-707 board achieved an input processing speed of 34.8 M input sample/s which is 3.3 times the input processing speed of ESN-FPGA accelerator for SISO system [1] where they manifested an input processing speed of 10.53 M input sample/s in maximum. In the case of the utilization of the proposed design, our accelerator gained 21.4% and 33% increase for LUT and FF respectively which is comparatively lower considering the high processing speed that was achieved with the architecture. Moreover, the design was able to save 50% in terms of BRAM memory space and also lowered 33.3% amount of use in DSP IP blocks in the implemented FPGA design which proved to be very cost-effective in terms of utilization, area, and power consumption.

VII. CONCLUSION

In this work, we develop a dedicated FPGA accelerator that implemented an ESN model for the data-intensive MIMO-OFDM symbol detection task for the first time. The design implemented a hardware-efficient ML model which can demonstrate significant performance in real-world scenarios. The ML algorithm displayed good performance with faster learning convergence in the case of limited data availability. Moreover, our accelerator architecture exhibited relatively low power consumption and memory space utilization for such a high throughput data-intensive communication task. We exploited the powerful DSP48E1 units efficiently which was able to surpass the DSP utilization of similar previous research that implemented high DSP-IP exploitation of FPGA for signal processing operations.

REFERENCES

[1] V. M. Gan, Y. Liang, L. Li, L. Liu, and Y. Yi, "A cost-efficient digital esn architecture on fpga for ofdm symbol detection," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 4, pp. 1–15, 2021.

[2] Z. Zhou, S. Jere, L. Zheng, and L. Liu, "Learning with knowledge of structure: A neural network-based approach for mimo-ofdm detection," in *2020 54th Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2020, pp. 22–26.

[3] Z. Zhou, L. Liu, and H.-H. Chang, "Learning for detection: Mimo-ofdm symbol detection through downlink pilots," *IEEE Transactions on Wireless Communications*, vol. 19, no. 6, pp. 3712–3726, 2020.

[4] B. V. Boas, W. Zirwas, and M. Haardt, "Deep-large: Higher-order svd and deep learning for model order selection in mimo ofdm systems," in *WSA & SCC 2023; 26th International ITG Workshop on Smart Antennas and 13th Conference on Systems, Communications, and Coding*. VDE, 2023, pp. 1–6.

[5] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International conference on machine learning*. PMLR, 2016, pp. 1225–1234.

[6] R. Shafin, L. Liu, J. Ashdown, J. Matyjas, M. Medley, B. Wysocki, and Y. Yi, "Realizing green symbol detection via reservoir computing: An energy-efficiency perspective," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[7] M. Wu, B. Yin, G. Wang, C. Dick, J. R. Cavallaro, and C. Studer, "Large-scale mimo detection for 3gpp lte: Algorithms and fpga implementations," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 5, pp. 916–929, 2014.

[8] O. Shears, K. Bai, L. Liu, and Y. Yi, "A hybrid fpga-asic delayed feedback reservoir system to enable spectrum sensing/sharing for low power iot devices iccad special session paper," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[9] S. Sesia, I. Toufik, and M. Baker, *LTE-the UMTS long term evolution: from theory to practice*. John Wiley & Sons, 2011.

[10] S. Jere, R. Safavinejad, and L. Liu, "Theoretical foundation and design guideline for reservoir computing-based mimo-ofdm symbol detection," *IEEE Transactions on Communications*, 2023.

[11] C. Lin, Y. Liang, and Y. Yi, "Fpga-based reservoir computing with optimized reservoir node architecture," in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2022, pp. 1–6.

[12] M. Lukoševičius, "A practical guide to applying echo state networks," *Neural Networks: Tricks of the Trade: Second Edition*, pp. 659–686, 2012.

[13] Y. Yi, Y. Liao, B. Wang, X. Fu, F. Shen, H. Hou, and L. Liu, "Fpga based spike-time dependent encoder and reservoir design in neuromorphic computing processors," *Microprocessors and Microsystems*, vol. 46, pp. 175–183, 2016.

[14] M. Bajger and A. Omondi, "Low-error, high-speed approximation of the sigmoid function for large fpga implementations," *Journal of Signal Processing Systems*, vol. 52, pp. 137–151, 2008.

[15] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "A survey on agent-based simulation using hardware accelerators," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–35, 2019.

[16] Y. Liang, L. Li, Y. Yi, and L. Liu, "Real-time machine learning for symbol detection in mimo-ofdm systems," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 2068–2077.