FPGA-based Reservoir Computing with Optimized Reservoir Node Architecture

Chunxiao Lin
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
chunxiaol@vt.edu

Yibin Liang
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
dliang@vt.edu

Yang Yi
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
yangyi8@vt.edu

2022 23rd International Symposium on Quality Electronic Design (ISQED) | 978-1-6654-9466-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISQED54688.2022.9806247

***Abstract*—Updating the state of reservoir nodes is one of the essential operations of reservoir computing (RC), which highly affects the system's performance. In an echo state network (ESN), one of the primary types of RC, the process of state renewal can be divided into two stages: multiplication of the weight matrix with the input-state vector and applying a nonlinear activation function on the sum of products. The weight matrix is typically large and sparse, providing opportunities for optimizing the matrix multiplication; the choices of activation functions may also affect hardware resource utilization. This paper introduces an optimized reservoir node architecture for FPGA-based RC systems. Specifically, we adopt the bit-serial matrix multiplier and direct spatial implementation of the weight matrix to fully exploit the sparseness property. The canonical signed digit representation is also employed to further optimize the multiplier logic. Furthermore, a hyperbolic tangent activation function is designed and optimized to maintain the nonlinearity of the neural network without affecting its accuracy. Compared with existing hardware ESN designs, our reservoir node architecture significantly reduces resource utilization while maintaining comparable performance.**

***Index Terms*—Reservoir computing, field-programmable gate array, echo state network, architecture**

I. Introduction

Artificial neural networks (ANN) are based on the concept of constructing a network from neurons or nodes, which are commonly denoted as black boxes with nonlinear activation functions. Signals can be transmitted between the neurons via complex connections, similar to the synapses in brains. In feedforward networks, neurons are fully or partially connected between adjacent layers. When arriving at the output nodes, the output signals will be reshaped based on the network connections and the activation function of internal neurons.

Feedforward neural networks without internal loops cannot solve some of the complex problems that require temporal information. Recurrent neural networks (RNNs) induce recurrent connections to solve the above difficulty. In RNNs, the input signals can be stored in the network for a longer time [1], and the states of neurons are dependent on both the current inputs and the previous ones. However, recurrence introduces difficulties like nonlinearity and high computational resource requirements.

Reservoir computing (RC) is a recently introduced framework for computation. It is derived from RNN but has better computation efficiency on many tasks that are deemed computationally hard, including time-series prediction [2], character recognition [3], or speech recognition [4], among others. By mapping the inputs to reservoir states in higher dimensions, an RC system only needs to train the connections from the internal reservoir to the output layer using simple linear regression techniques. Therefore, the training of RNN is highly simplified. The implementation of reservoir computing varies, among which the three common types are echo state network (ESN) [8], liquid state machine (LSM) [5], and delayed feedback reservoir (DFR) [6], [7].

Modern ESN system designs require less costly, more energy-efficient, and fast platforms. Although the training procedure of RC is highly improved, the computational cost of ESN is still significant due to a large number of multiply-and-accumulate (MAC) operations and nonlinear function mappings. Field-programmable gate arrays (FPGAs) are reconfigurable computer chips consisting of different types of flexibly interconnected programmable blocks. Compared with application-specific integrated circuits (ASIC), FPGAs require much less non-recurring engineering costs. Unlike CPUs or GPUs with fixed architecture for general use cases, FPGAs enable hardware implementation with specific architecture reconfigurable to target applications. Moreover, FPGA can utilize extensive parallelism to increase computational speed.

This paper proposes an optimized reservoir node architecture for the ESN system. Considering that the weight matrices are large, sparse, and unchanged during the training and inference operations, we adopt the bit-serial vector-matrix multiplier for the multiply-and-accumulate (MAC) operation and optimized implementation of the weight matrices in the hardware. As for the activation function, a hyperbolic tangent function is chosen and further designed and optimized for low resource utilization. This novel design can significantly reduce resource utilization without introducing performance losses compared to prior FPGA-based ESN hardware designs.

The structure of the paper is as follows: In Section II, we introduce the background of reservoir computing, including the ESN system model and state-of-the-art FPGA architecture

Authorized licensed use limited to: to IEEExplore provided by University Libraries | Virginia Tech. Downloaded on December 15,2024 at 20:53:39 UTC from IEEE Xplore. Restrictions apply.

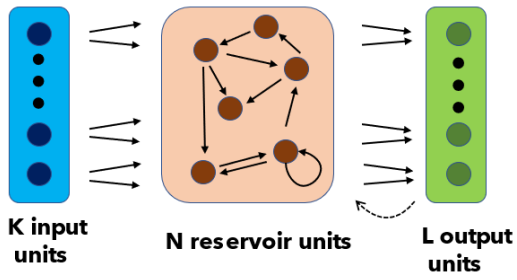


Fig. 1. The architecture of a typical echo state network, with K inputs, N reservoir nodes, and L outputs

optimizations. Section III reviews the bit-serial computation mechanism and describes the architecture of the bit-serial multiplier and the activation function. Section IV compares the device utilization and throughput of the ESN system using the introduced reservoir node architecture with other implementations [18]. Finally, Section V summarizes this paper and discusses future work.

II. Background

A. Reservior Computing

The concept of reservoir computing is mainly inspired by the information processing procedure of biological brains. The input signals activate the neurons and generate different patterns of neuronal activities. Reservoir computing generalizes several traditional neural networks, such as echo state networks and liquid state machines. The similarities of these models are that they all focus on dynamic systems to solve temporal problems. Reservoir computing successfully solved one critical drawback of RNNs: the high computational training cost.

In RC systems, a reservoir is an extensive network consisting of reservoir neurons (or reservoir nodes) as the critical part of the system. It receives input signals from the input layer and nonlinearly transforms them into a high-dimensional space. This function requires the reservoir to contain enough neurons with recurrent interconnections. The input history information and the states of reservoir nodes are stored in the network to update current states and states in the future. Only the connections between the reservoir and the final output layer need to be trained. Since only linear regression strategies are needed for training the output weights, the training of recurrent networks becomes feasible and costs less [9].

B. Echo State Network

Echo state network was first proposed by Jaeger in [8] to improve the efficiency of the training of RNNs. It has been further developed in the following decades [10]–[12]. Figure 1 shows the typical structure of ESN. There are three components in the model which corresponds to the three parts of an RC system: the input layer receives the input series; the reservoir layer finishes the input mapping, updates and store the states of reservoir nodes; the output layer generates the transformed signals for prediction and training.

As is shown in Figure 1, $\boldsymbol{u}(n) = (u_1(n), u_2(n), ..., u_K(n))$ denotes the input vector at timestep n, and $\boldsymbol{y}(n) = (y_1(n), y_2(n), ..., y_L(n))$ denotes the output vector. The reservoir states stand for the output of each reservoir node, which are written as $\boldsymbol{x}(n) = (x_1(n), x_2(n), ..., x_N(n))$. The reservoir system can be described using the following equations [13]:

$$x(n+1) = f(W^{in}u(n+1) + W^{x}x(n) + W^{fb}y(n)), \quad (1)$$

$$y(n+1) = f^{out}(W^{out}[u(n+1), x(n+1), y(n)]). \quad (2)$$

Equations 1 and 2 represent the process of calculating the current states and the current outputs, where W^{in}, W^{x}, W^{fb}, and W^{out} denote the input, reservoir, feedback, and output weight matrix, respectively; f is the activation function used in reservoir nodes, and f^{out} is the activation function for the output layer.

To simplify the notation, an extended state vector $\boldsymbol{z}(n)$ is introduced to represent a concatenation of $\{\boldsymbol{x}(n); \boldsymbol{u}(n); \boldsymbol{y}(n-1)\}$. The weight matrices W^{in}, W^{x}, W^{fb} can be also concatenated and replaced by a extended weight matrix W^{ex}. Then, equation 1 and 2 can be simplified as follows:

$$x(n+1) = f(W^{ex}z(n+1)), \quad (3)$$

$$y(n+1) = f^{out}(W^{out}z(n+1)). \quad (4)$$

Only the output weights need to be updated when training an ESN. Linear regression algorithms, such as least mean squares and recurrent least squares, can be applied to train the output weights and minimize the training error.

C. State-of-the-art RC System Implementation

In reservoir computing, FPGA is still one of the significant research and development platforms. The bit-level configurability enables more flexible architectures than CPUs or GPUs. On the other hand, ASIC chips are a different category of digital systems, with similar advantages like flexibility, reconfigurability, and parallelism. The analog circuit implementations of RC, especially the delayed feedback reservoir (DFR), have received more and more attention due to the simple reservoir structure. Previously, [14] presented the implementation of a complete analog DFR computing system. However, compared with FPGA, ASIC design of RC system are more limited in the time-to-market and the cost of the non-repeating technical design.

There have been ongoing investigations of the FPGA design of the RC system since the concept of RC was proposed. In [15], David V. et al. first discussed an RC system with analog neurons on FPGA. The new type of neurons communicates using stochastic bitstreams instead of fixed values, which simplifies the implementation of arithmetic operations in RC updating. A liquid-state-machine-based hardware implementation for real-time speech recognition is realized by Benjamin et al. in [16]. Moreover, in [17], the ESN architecture can be fully implemented in the FPGA platform without software assistance, including the training process. Recently, a cost-efficient ESN architecture on FPGA has been proposed in [18], which optimizes the utilization of DSP units inside the FPGA.

Authorized licensed use limited to: to IEEExplore provided by University Libraries | Virginia Tech. Downloaded on December 15,2024 at 20:53:39 UTC from IEEE Xplore. Restrictions apply.

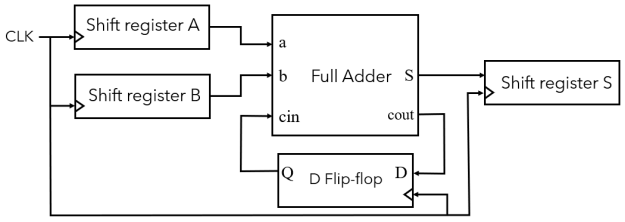


Fig. 2. The structure of a bit-serial adder

In contrast, this work focuses on optimizing the multipliers for MAC operation, considering the sparse characteristic of the matrices in the calculation.

III. Proposed Architecture of ESN Reservoir Neurons

A. Bit-serial Matrix-vector Multipliers

As discussed in Section II, the reservoir nodes were updated in two steps: the MAC operation and the subsequent nonlinear activation. Since the reservoir is supposed to be sparse, a substantial proportion of the connection weights within a reservoir are zeros. Therefore, the weight matrix in the MAC operation should be large, sparse, and fixed. This allows optimizing the multiplication implementation through constant propagation, i.e., simplifying the multiplication with zero. Considering that not only the value of a weight but also the individual bits of the value can be zeros, we adopt the bit-serial multiplier for the MAC operation to further reduce the number of bit multipliers.

In this work, the embedded multipliers or the digital signal processing units in FPGA are not considered. The bit-serial multipliers are implemented by re-purposing the look-up tables in FPGA into basic devices like shift registers and utilizing the registers inside. The details of the architecture of the bit-serial devices are described below.

1) Bit-serial adder: The bit-serial adder is the essential component of bit-serial designs. As shown in Figure 2, a bit-serial adder consists of a full adder, a D flip-flop, and three shift registers. The two operands are stored in two right-shift registers A and B. A pair of bits are sent to the full adder for calculation at each clock cycle. The carry bit is temporarily kept in a D flip-flop as the carry-in bit of the next addition operation, while the sum of the current pair of inputs is sent to the output shift register.

The bit-serial subtractor can be easily created from the bit-serial adder. Similar to the subtraction of signed two's complement numbers, taking the case $a-b$ as an example, the operation generates the same output as $a+(-b)+1$, which can be implemented by adding an inverter before the shift register of b and set the initial value of the carry-in bit to 1.

2) Single-bit Vector Multiplier: $\boldsymbol{a} = (a_1, a_2, ..., a_N)$ and $\boldsymbol{b} = (b_1, b_2, ..., b_N)$ are two vectors of length N, with a length of 1 bit for each element. The dot product of $\boldsymbol{a}$ and $\boldsymbol{b}$ can be calculated as

$$S = \sum_{i=1}^{i=N} a_i * b_i. \tag{5}$$

In this MAC operation, the multiplication between bits can be implemented using only the AND gates, and the products will be accumulated by a tree of bit-serial adders.

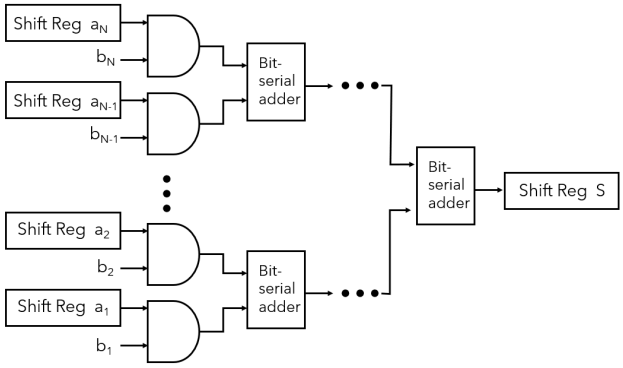


Fig. 3. The design of multi-bit by single-bit multiplier

3) Multi-bit by Single-bit Multiplier: Suppose the bit width of elements in $\boldsymbol{a}$ is k $(k > 1)$ and the elements in $\boldsymbol{b}$ remain 1-bit wide. Each multi-bit $a_i(\overline{a_{i,k-1}a_{i,k-2}...a_{i,0}})$ is stored in a shift register and sent to an AND gate to multiply with b_i, one bit per clock cycle from LSB to MSB. The products are accumulated by a tree of bit-serial adders, and the final result is stored in another shift register. The calculation process is described in the following equations:

$$S = \sum_{i=1}^{N} a_i * b_i \tag{6}$$

$$= \sum_{i=1}^{N} (\overline{a_{i,k-1}a_{i,k-2}...a_{i,0}}) * b_i \tag{7}$$

$$= \sum_{i=1}^{N} (\sum_{j=0}^{k-1} 2^j a_{i,j}) * b_i \tag{8}$$

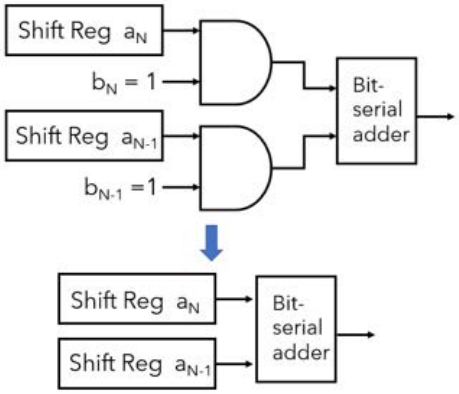
$$= \sum_{i=1}^{N} \sum_{j=0}^{k-1} 2^j a_{i,j} * b_i \tag{9}$$

$$= \sum_{j=0}^{k-1} (\sum_{i=1}^{N} 2^j a_{i,j} * b_i). \tag{10}$$

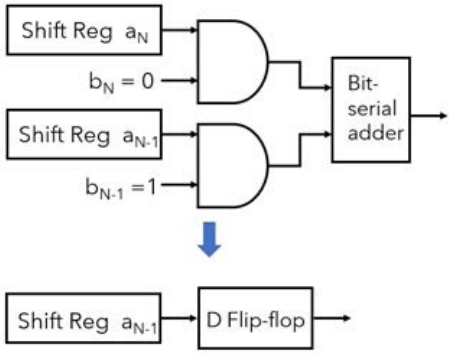
In equation 10, the jth bit of the k-bit result S is $\sum_{i=1}^{N} a_{i,j} * b_i$, which can be implemented as Figure 3.

Considering that the weight matrix is fixed in the ESN design, the vector $\boldsymbol{b}$ can be regarded as constant. Therefore, the element b_i will be either 0 or 1, leading to the logic optimization on the tree of bit-serial adders. If b_i is fixed to 1, one of the inputs of the AND gate is always 1, and the output will be the direct propagation of a_i. In this case, the AND gate can be removed without affecting the logic of the circuit, i.e.,

Authorized licensed use limited to: to IEEExplore provided by University Libraries | Virginia Tech. Downloaded on December 15,2024 at 20:53:39 UTC from IEEE Xplore. Restrictions apply.



(a) First principle for cases where bi = 1



(b) Second principle for cases where bi = 0

Fig. 4. Two optimization principles of the multi-bit by single-bit multiplier

the shift register of a_i can be connected to the bit-serial adder directly. If b_i is fixed to 0, the AND gate will only transmit a zero to the adder. In this case, the adder can be replaced by a D flip-flop to store the output of the other AND gate. In summary, the original circuit can be significantly optimized on the cost with these two principles, as illustrated in Figure 4.

4) Multi-bit Vector Multiplier: For multi-bit by multi-bit vector multiplier, the elements in vector $\boldsymbol{b}$ will be more than 1 bit wide. Assuming the bit width of them is t $(t > 1)$, the result of multiplication of $\boldsymbol{a}$ and $\boldsymbol{b}$ can be calculated as follows:

$$S = \sum_{i=1}^{N} a_i * b_i \tag{11}$$

$$= \sum_{i=1}^{N} (\overline{a_{i,k-1}a_{i,k-2}...a_{i,0}}) * (\overline{b_{i,t-1}b_{i,t-2}...b_{i,0}}) \tag{12}$$

$$= \sum_{i=1}^{N} ((\sum_{j=0}^{k-1} 2^j a_{i,j}) * (\sum_{m=0}^{t-1} 2^m b_{i,m})) \tag{13}$$

$$= \sum_{m=1}^{t-1} 2^m (\sum_{j=0}^{k-1} (\sum_{i=1}^{N} 2^j a_{i,j} * b_{i,m}). \tag{14}$$

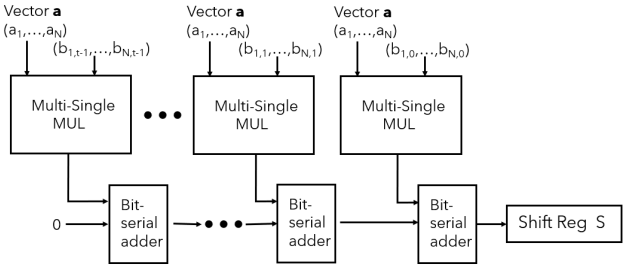


Fig. 5. The design of the multi-bit multiplier

In equation 14, the summation $\sum_{j=0}^{k-1}(\sum_{i=1}^{N} 2^j a_{i,j} * b_{i,m})$ can be implemented as the multi-bit by single-bit multiplier (multi-single multiplier). To combine these products with different weights, we adopt a chain of bit-serial adders to connect all the outputs of the multi-single multipliers and store the final result into a shift register. With one clock delay between adjacent multipliers, the products are automatically timed by their weights. The design of the multi-bit multiplier is shown in Figure 5.

5) Vector-matrix Multiplier: With the multi-bit vector multiplier, we can now deal with the MAC operations in the reservoir nodes, which is the multiplication between an input vector and a weight matrix. In equation 3, the dimensions of the extended weight matrix W^{ex} are $N \times (K+N)$, and the extended input vector $\boldsymbol{z}(t)$ is composed of $(K+N)$ elements. To generate the $(N \times 1)$ product, in which each row is used to produce the new state of one reservoir node, we can calculate the elements separately. Each row of the W^{ex} is a vector with dimensions $1 \times (K+N)$. We multiply the vectors with $\boldsymbol{z}(t)$ using multi-bit vector multipliers, and the results make up the output vector we want. The architecture of the final multiplier design is depicted in Figure 6. The calculation process can be described by the following equations:

$$output = \begin{pmatrix} w_{0,0} & \cdots & w_{0,K+N-1} \\ \vdots & \ddots & \vdots \\ w_{N-2,0} & \cdots & w_{N-2,K+N-1} \\ w_{N-1,0} & \cdots & w_{N-1,K+N-1} \end{pmatrix} \begin{pmatrix} z_0(t) \\ \vdots \\ z_{K+N-2}(t) \\ z_{K+N-1}(t) \end{pmatrix}$$

$$= \begin{pmatrix} row\_vector_0 \\ \vdots \\ row\_vector_{N-2} \\ row\_vector_{N-1} \end{pmatrix} input\_vector^T$$

$$= \begin{pmatrix} row\_vector_0 * input\_vector \\ \vdots \\ row\_vector_{N-2} * input\_vector \\ row\_vector_{N-1} * input\_vector \end{pmatrix}$$

B. Canonical Signed Digit

For the multi-bit by single-bit multiplier, its operation can be regarded as the summing of the a_i selected by b_i. This design works with both signed and unsigned a, which represents the inputs of the RC system. However, only the unsigned weights

Authorized licensed use limited to: to IEEExplore provided by University Libraries | Virginia Tech. Downloaded on December 15,2024 at 20:53:39 UTC from IEEE Xplore. Restrictions apply.

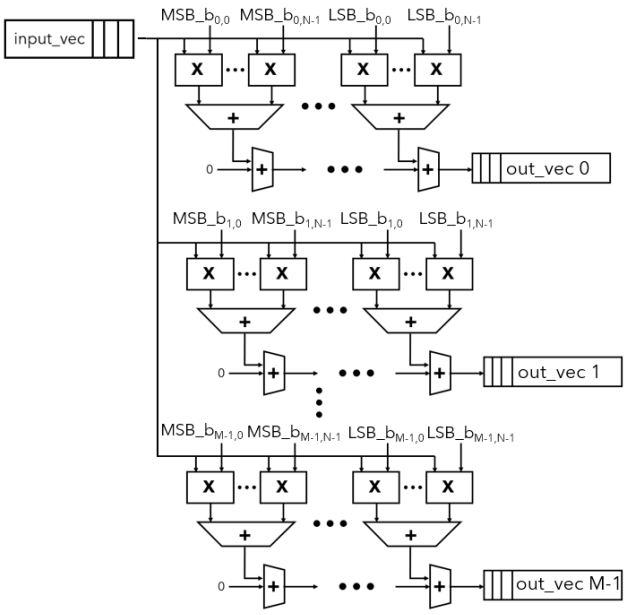


Fig. 6. Overall architecture of the final vector-matrix multiplier

b is allowed. One solution is to encode the signed weights into two pairs of unsigned weights, multiply the inputs with them separately, and subtract the two results to get the final results. The method we choose is canonical signed digit (CSD).

One advantage of the CSD representation is that it makes the digital multipliers more efficient. Suppose that we calculate the multiplication between x and a binary number $y = 11110$. The standard process is as

$$product = 2^4x + 2^3x + 2^2x + 2^1x, \tag{15}$$

which contains four shift operations and three additional operations. However, if we replace the binary number y with a subtraction like $y = 11110 = 100000 - 000010$, equation 15 can be simplified as

$$product = 2^5x - 2^1x, \tag{16}$$

for which only two shifts and a subtraction being needed. Thus, the CSD representation minimizes the number of non-zero digits. In this way, we are able to optimize the power of the multiplications using CSD numbers.

Transforming unsigned binary numbers to CSD numbers is simple. We use three digits 1, 0, -1 to build the new format and search the sequence of unsigned binary numbers. If there are subsequences with continuous ones like '0111', we replace it with a new sequence '100(-1)' to reduce the proportion of ones. If the highest several bits are '111', we need to extend the number with a 0 at the head and then use the transforming principle. In this way, the generated CSD numbers will be one bit wider than the original binary numbers. Furthermore, there can be more than one iteration since the replacement may introduce a new subsequence of continuous ones.

To apply the CSD presentation to the vector-matrix multiplication, we need to separate the CSD number into two unsigned numbers that can be recognized for calculation. Taking '1011' for example, we can translate it into the CSD format '10(-1)0(-1)' with two iterations. Here the value of the number is

$$1 \times 2^4 + (-1) \times 2^2 + (-1) \times 2^0. \tag{17}$$

To maintain the same value, we use the subtraction of two 5-bit unsigned numbers '10000' and '00101' to substitute the CSD number with three types of digits. And the result is the same as the original value.

As for signed binary numbers, we need to slightly modify the transferring process. If the number is negative, we need to flip the sign bit of the first unsigned number that is separated from the CSD number.

C. Nonlinear Activation Function: Hyperbolic Tangent

The nonlinear activation function is the other step of the reservoir updating besides the MAC operation, which also influences the cost of the hardware implementation. In many cases, nonlinear functions like hyperbolic tangent ($tanh$) [18] are applied as the activation function of the reservoir nodes to increase the nonlinearity. In our design, we also choose the $tanh$ function for the sake of accuracy on the application of OFDM symbol detection, the details of which are described in [18]. However, compared with the design in [18], we can use simple DSP units instead of large DSP groups to realize the activation function, which reduces the utilization of DSPs to a large degree.

IV. Experimental Results

We apply the new architecture of the reservoir node to an ESN system, adopting the bit-serial multipliers, the CSD algorithm, and the simplified hyperbolic tangent activation function. And then test the ESN architecture on a symbol detection task in an orthogonal frequency division multiplexing (OFDM) system, which is fully described in [18]. The design is written in SystemVerilog and synthesized using Xilinx Vivado 2019.1.3. Since the bit-serial multiplier is the essential part of our design, we first test the function and the performance of it with a direct input vector and weight matrix. The result is generated with a delay of 44 clocks, which is related to the bitwidth of the input, the bitwidth of weights ,and the length of the input vector (which decides the depth of the adder tree within the multiplier).

After that, based on our previous work in [18], we synthesize the new ESN structure with optimized reservoir nodes on the Virtex-7 VC707 Evaluation Platform and compare it with the original DSP-based ESN. The comparison of the resource utilization between them is presented in Table II, where we can see that the new ESN significantly reduces the resource needed for implementation. While optimizing the utilization of DSP, the number of look-up tables and flip-flops are also reduced to a smaller quantity, which is attributed to the adoption of the new structure of multiplier and the simplification of the hyperbolic tangent activation function. Moreover, we also compare the throughput of these two designs, and the result is shown in Table III. For each testing case, there are 4 inputs

Authorized licensed use limited to: to IEEExplore provided by University Libraries | Virginia Tech. Downloaded on December 15,2024 at 20:53:39 UTC from IEEE Xplore. Restrictions apply.

of 16-bit width. For the design in [18], the time interval of input streams is 0.46 us, which is increased to 0.83 us in the proposed design. The 44.5% loss of throughput in ESN is acceptable, considering the significant improvement in hardware cost.

TABLE I
PARAMETERS OF ESN FOR OFDM SYMBOL DETECTION

	Symbol Detection
Number of input nodes	4
Number of reservoir nodes	16
Number of output nodes	2
Range of inputs	(-1, 1)
Range of outputs	(-10,10)
Range of learned weights	(-5000,5000)
Bit width of outputs of activation function	20 bits

TABLE II
RESOURCE UTILIZATION COMPARISON BETWEEN DSP-BASED AND PROPOSED ARCHITECTURES

	ESN with DSP-based neurons	ESN with neurons of proposed architecture
LUT	11011 (3.63%)	2133 (0.7%)
FF	7001 (1.15%)	5978 (0.98%)
BRAM	12 (1.17%)	0
DSP	162 (5.79%)	16 (0.57%)
IO	389 (55.57%)	389 (55.57%)
BUFG	1 (3.13%)	1 (3.13%)

TABLE III
THROUGHPUT COMPARISON BETWEEN THE TWO ESN DESIGNS

	ESN with DSP-based neurons	ESN with neurons of proposed architecture
Throughput	64bits / 0.46us	64bits / 0.83us

V. SUMMARY AND CONCLUSIONS

This paper introduces an optimized architecture of reservoir nodes for ESN systems. The corresponding FPGA implementation is verified with an OFDM symbol detection application. The optimization of the reservoir node architecture includes a bit-serial multiplier, a canonical signed digit representation, and a simplified hyperbolic tangent activation function, which aims at reducing the device utilization without significant loss in accuracy and throughput. Compared with previous DSP-based ESN design, the proposed ESN architecture eliminates most DSP blocks and reduces the LUT utilization by 80.7% and FF by 14.8%. The costs of the above improvements are the throughput lost due to the bit-serial operation mechanism and the speed decrease of MAC compared with built-in DSP. Our work presents a successful design trade-off between resource utilization and performance to meet specific application requirements.

ACKNOWLEDGMENT

This work was supported in part by the U.S. National Science Foundation (NSF) under Grants CCF-1750450, ECCS-1731928, and CCF-1937487.

REFERENCES

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp.85–117, 2015.

[2] wyffels, Francis and Schrauwen, Benjamin and Stroobandt, Dirk, "Using reservoir computing in a decomposition approach for time series prediction," in Proceedings of ESTSP 2008 European Symposium on Time Series Prediction, Lendasse, Amaury, Ed. Multiprint Oy/Otamedia, 2008, pp.149–158.

[3] Y. Jin, Q. Zhao, H. Yin, and H. Yue, "Handwritten numeral recognition utilizing reservoir computing subject to optoelectronic feedback," in 2015 11th International Conference on Natural Computation (ICNC). IEEE, 2015, pp. 1165–1169.

[4] M. Skowronski and J. Harris, "2007 special issue: Automatic speech recognition using a predictive echo state network classifier," Neural networks : the official journal of the International Neural Network Society, vol. 20, pp.414–23, 05 2007.

[5] W. Maass, T. Natschl ager, and H. Markram, "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations," Neural Computation, vol. 14, no. 11, pp.2531–2560, 11 2002.

[6] L. Grigoryeva, J. Henriques, L. Larger, and J.-P. Ortega, "Optimal nonlinear information processing capacity in delay-based reservoir computers," Scientific Reports, vol. 5, no. 1, p. 12858, Sep 2015.

[7] S. Ort in and L. Pesquera, "Reservoir computing with an ensemble of time-delay reservoirs," Cognitive Computation, vol. 9, 06 2017.

[8] H. Jaeger, "The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'," Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, vol. 148, 01 2001.

[9] L. Appeltant, M. C. Soriano, G. V. der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," Nature Communications, vol. 2, 2011.

[10] H. Jaeger, "Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the echo state network approach," GMD-Forschungszentrum Informationstechnik, 2002., vol. 5, 01 2002.

[11] H. Jaeger, "Discovering multiscale dynamical features with hierarchical echo state networks," 2007.

[12] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," Frontiers in Neuroscience, vol. 11, 2017.

[13] H. Paugam-Moisy, "Spiking neuron networks a survey," 2006.

[14] K. Bai and Y. Yi, "Dfr: An energy-efficient analog delay feedback reservoir computing system for brain-inspired computing," J. Emerg. Technol. Comput. Syst., vol. 14, no. 4, dec 2018.

[15] D. Verstraeten, B. Schrauwen, and D. Stroobandt, "Reservoir computing with stochastic bitstream neurons," in Proceedings of the 16th annual Prorisc workshop, 2005, pp. 454–459.

[16] B. Schrauwen, M. D'Haene, D. Verstraeten, and J. V. Campenhout, "Compact hardware liquid state machines on fpga for real-time speech recognition," Neural Netw., vol. 21, no. 2, p. 511–523, mar 2008.

[17] 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.

[18] V. M. Gan, Y. Liang, L. Li, L. Liu, and Y. Yi, "A cost-efficient digital esn architecture on fpga for ofdm symbol detection," J. Emerg. Technol. Comput. Syst., vol. 17, no. 4, jun 2021.

Authorized licensed use limited to: to IEEExplore provided by University Libraries | Virginia Tech. Downloaded on December 15,2024 at 20:53:39 UTC from IEEE Xplore. Restrictions apply.