LUTNet-RC: Look-Up Tables Networks for Reservoir Computing on an FPGA

Kanta Yoshioka*, Yuichiro Tanaka†, and Hakaru Tamukoh*†
*Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Japan
†Research Center for Neuromorphic AI Hardware, Kyushu Institute of Technology, Japan
Email: yoshioka.kanta986@mail.kyutech.jp

***Abstract*—We propose look-up tables networks-based reservoir computing (LUTNet-RC). This work is the first trial of applying LUTNets to RC. LUTNet-RC consists of a LUT-based reservoir layer and a non-LUT-based output layer. LUTNets have disadvantages such as limited sparse connectivity and weights cannot be changed after implementation. However, when applied to a reservoir layer of RC (LUT-based reservoir layer), these disadvantages are eliminated, because this layer works with sparse connectivity and the weights are fixed, so only the advantage of small circuit resources is obtained. For the LUT-based reservoir layer, we propose and model a multi-bit weight reservoir, modifying the conventional binarized reservoir to improve calculation accuracy. In the case of LUTNets, the proposed multi-bit weight reservoir can be implemented without the increase in utilized circuit resources because LUTNets focus only on the input-output relationship on neurons. Additionally, we propose a speed-up method in the output layer with time division calculation, which compares the current network state with previous states and then calculates only status-changed neurons. As a result, we implement a LUTNet-RC with 1500 reservoir neurons on a field-programmable gate array (KR260) running at 100MHz. The utilized circuit resources are dominated by LUTs, which use approximately 26% of the total amount of LUTs. The LUTNet-RC can infer more than 10^6 data per second. We also verify the LUTNet-RC performance using non-linear auto-regressive moving average 10 (NARMA10) and the performance is comparable to conventional works. We conclude that the LUTNet-RC is one of the highest-performance RC on an FPGA.**

***Index Terms*—neural networks, reservoir computing, field programmable gate array**

I. Introduction

With the increasing requirements for high-performance automated vehicles, smartphones, and home-service robots, an increasing need exists for real-time data processing in edge computing [1], [2]. Reservoir computing (RC) [3] has attracted attention for real-time data processing in edge environments because of its low training and inferring cost and high accuracy comparable to multi-layer recurrent neural networks (RNNs), with a single-layer sparse random network (reservoir layer) [4], [5]. RC performance depends on the reservoir layer topology, often fine-tuned for different tasks. In graphics processing unit (GPU)-implemented RCs, the topology is realized by changing the weight matrix, while in field-programmable gate array (FPGA)-implemented RCs, the topology is realized directly as circuits. Then, RCs can be implemented without huge complex matrix circuits like GPUs and enhance parallelism using simple circuits. Therefore, FPGA implementation of RC has advantages in realizing low-power, high-performance, and flexible edge systems, and is progressing [6]. However, because of the limitation of circuit resources, many FPGA-implemented RCs have limited the number of arithmetic units and reduced bit precision [6].

This work focuses on look-up tables networks (LUTNets) [7], one of the binarized neural networks (BNNs) [8], which is more specialized for FPGA implementation than general BNNs. LUTNets are neural networks constructed using neurons oriented to LUT elements on FPGAs, focusing only on the input-output relationship of neurons. LUTNets can operate with smaller circuit resources and higher recognition accuracy than general BNNs implemented on FPGAs [9] [10].

LUTNets have disadvantages such as limited sparse connectivity, and weights cannot be changed after implementation. However, when applied to a reservoir layer of RC, these disadvantages are eliminated because this layer works with sparse connectivity, and the weights are fixed; thus, only the advantage of small circuit resources is obtained.

From this good complementarity between RC and LUTNets, we propose LUTNet-based RC (LUTNet-RC), which consists of a LUT-based reservoir layer and a non-LUT-based output layer. We implemented LUTNet-RC on an FPGA and evaluated its computational accuracy and speed and circuit resources. The main contributions of this work are summarized as follows.

- We discovered the good complementarity between RC and LUTNet and proposed LUTNet-RC, which is an application of LUTNet to RC.
- We proposed a novel RC model suited for implementation with LUTNets. The circuit program for FPGA implementation was written in Verilog HDL and was automatically generated by software programs in Python from set parameters.
- We proposed a novel hardware-oriented algorithm to accelerate time-division multiply-accumulations. The acceleration method is suited for models with many neuron changes such as LUTNet-RC.
- We implemented the proposed LUTNet-RC on an FPGA. The calculation accuracy of the LUTNet-RC was comparable to that of the conventional RC on FPGAs, while the calcu-

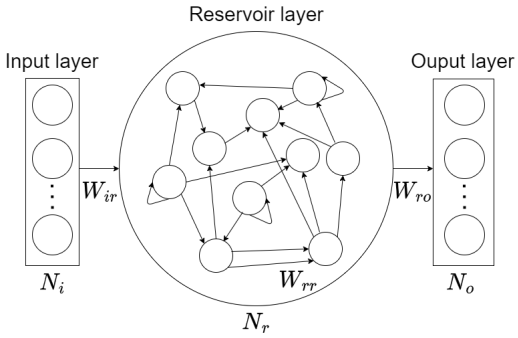


Fig. 1. Reservoir computing model

lation speed and circuit resources were greatly improved. The LUTNet-RC implemented in this work is one of the highest-performance RC on an FPGA.

II. Related works

A. Reservoir computing

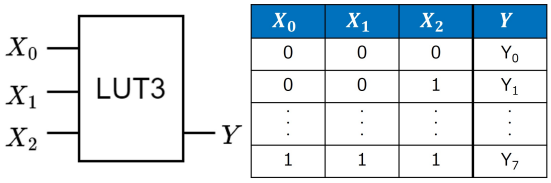
Reservoir computing (RC), a type of RNN, consists of three layers: an input layer, a reservoir layer, and an output layer, as shown in Fig. 1 [3]. The input, reservoir, and output layers contain N_i, N_r, and N_o neurons, respectively. W_{ir}, W_{rr}, and W_{ro} denote the weights between the input and reservoir layers, inside the reservoir layer, and between the reservoir and output layers, respectively. Although RC has a simple structure, its performance is comparable to that of deep-learning (DL)-based RNNs with many layers [4] [5]. In addition, the weights W_{ir} and W_{rr} are fixed, and only W_{ro} be trained, so the training cost is lower than DL-based RNNs.

In RC, memory capacity and nonlinearity are important indices, but a trade-off exists between them [11], [12]. Therefore, various RC models with characteristics of memory capacity and nonlinearity have been proposed, such as echo state networks (ESNs) [3], liquid state machines (LSMs) [13], RC based on chaotic Boltzmann machines (CBM-RCs) [14], RC based on pulse-coupled phase oscillators (PCPO-RCs) [15], and RC based on chaotic neural networks (ChNN-RCs) [16].

RC has attracted attention for real-time data processing in edge environments because of its performance and low training cost, as described previously. FPGA implementations have been progressing because of their low power consumption and flexible implementation [6]. However, owing to the limited circuit resources in FPGAs, RC must be implemented with few arithmetic operators and low bit precision, and the calculation accuracy and speed of RC circuits are constrained.

B. LUTNet

BNNs are quantized neural network models in which the weights and outputs of each neuron in the neural network are binarized [8]. Therefore, the amount of memory required to store weights is reduced, and the multiplier can be implemented with simple logic elements, greatly reducing circuit



X_0	X_1	X_2	Y
0	0	0	Y_0
0	0	1	Y_1
⋮	⋮	⋮	⋮
1	1	1	Y_7

Fig. 2. Three-inpput one-output look-up table

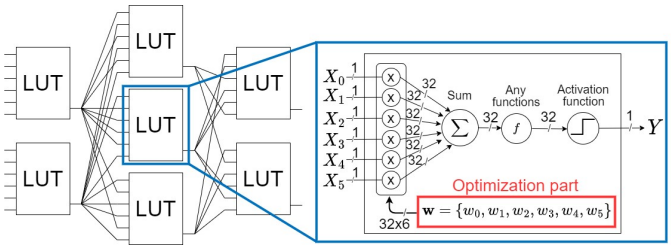


Fig. 3. Neural network with LUT-based neurons

resources when implemented in FPGAs [10]. LUTNets are BNNs, but unlike general BNNs, LUTNets are designed for FPGA implementation [7].

A look-up table (LUT) is a logic block in FPGAs. In the case of a three-input one-output LUT (LUT3), as shown in Fig. 2, the structure is similar to a truth table in which one output Y is determined by the three inputs X_0, X_1, and X_2. In FPGAs, any operation can be realized by changing the connection between LUTs and the output table $\mathbf{Y}$ in each LUT.

One LUTNet implementation method defines a neuron model based on the LUT and constructs neural networks using it, as shown in Fig. 3 [7], [9], [17], [18]. The neural network is trained by optimizing the weights, similar to normal neural networks. Then, the input-output relationship of each neuron is mapped onto a LUT. Since only the input-output relationship of each neuron is focused on, the bit precision of the weights does not affect the circuit resources of the LUTNet when implemented on FPGAs. LUTNets have been applied to image recognition tasks and operate with smaller circuit resources and higher accuracy than general BNNs [9], [10].

Another implementation method involves constructing networks by directly connecting the truth tables and changing the output table Y directly for training [19], [20]. This method mainly contributes to the theory of neural networks and is expected to be able to solve the mysteries related to the generalization and memorization of neural networks.

C. Differential multiply-accumulation

A differential multiply-accumulation (DMACC) is a scheme for neural networks, which reduces the circuit resource utilization and calculation time of multiply accumulations as shown in Eq. (1) [21], [22].

$$z_i[t] = b_i + \sum_{j=1}^{N} s_j[t] * w_{i,j}, \tag{1}$$

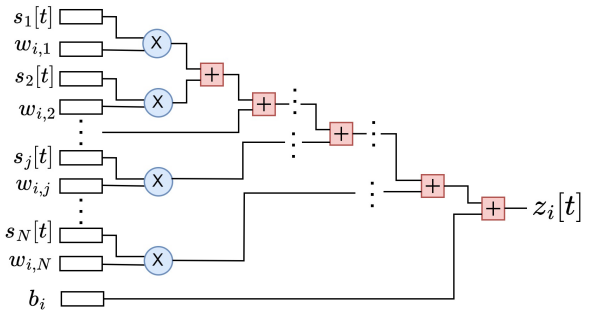


Fig. 4. All-parallel implementation of $z_i[t]$ calculation circuit

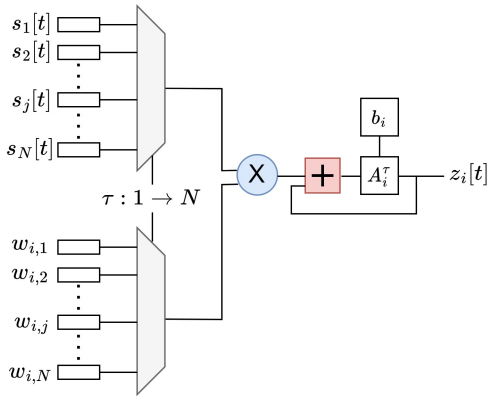


Fig. 5. Time-division implementation of $z_i[t]$ calculation circuit

where $z_i[t]$, b_i, $s_j[t] \in \{0, 1\}$, w_{ij}, and N represent output of ith neuron at time t, bias of ith neuron, external state of jth neuron at time t, the weights between ith and jth neurons, and the number of neurons, respectively.

One of the approaches to accelerating the multiply-accumulation like Eq. (1) is implementing all multiply-accumulation operators in parallel, as shown in Fig. 4. However, the number of multiply-accumulation operators increases in proportion to the square of the number of neurons. Therefore, it is difficult to implement all multiply-accumulate operators in parallel.

One technique that is often used is time-division calculation. The circuit that introduces the time-division technique is shown in Fig. 5. In simple time-division calculations (Simple-TDC), the output value of ith neuron $z_i[t]$ is defined by Eqs. (2) and (3), where $\tau \in [1, N]$ and A_i^τ represent an iterator of the time-division and an accumulated value at time τ, respectively. The circuit in Fig. 5 holds the accumulated value in a register and adds the input values to it.

$$z_i[t] = A_i^N, \tag{2}$$

$$A_i^\tau = A_i^{\tau-1} + s_\tau[t] * w_{i,\tau} (\tau \neq 0), \ A_i^0 = b_i. \tag{3}$$

Using Simple-TDC, only N multiply-accumulation operators are implemented on an FPGA, and z_i in Eq. (1) is

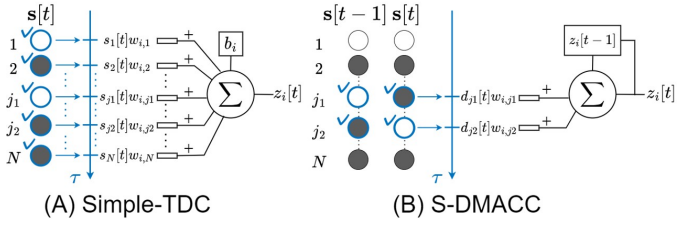


Fig. 6. Conventional time-division calculation methods

calculated by using them N times. Therefore, the number of multiply-accumulation operators increases in proportion to the number of neurons.

However, for circuits that introduce Simple-TDC in Fig. 6(A), the calculation time increases significantly. For example, for $N = 1000$, the multiply-accumulation operators must be used 1000 times to calculate $z_i[t]$ in Eq. (1), and the calculation time increases proportionally with the number of neurons.

Because of these disadvantages, a single-DMACC (S-DMACC) method was proposed [21], [22], which is an improved hardware-oriented algorithm of the conventional Simple-TDC described previously, to reduce the increase in calculation time. S-DMACC, which compares the current network state $\mathbf{s}[t]$ with the previous network state $\mathbf{s}[t-1]$ and calculates only the changed neurons using the value calculated one time ago $z_i[t-1]$, as shown in Fig. 6(B). The calculation is described in the following Eqs. (4), (5), and (6), where $d_j[t]$ represents the difference between the previous output of the j-th neuron $s_j[t-1]$ and the present output for $s_j[t]$.

$$z_i[0] = b_i + \sum_{j=1}^{N} s_j[0] * w_{i,j}, \tag{4}$$

$$z_i[t] = z_i[t-1] + \sum_{s_j[t-1] \neq s_j[t]} d_j[t] * w_{i,j}, \tag{5}$$

$$d_j[t] = \begin{cases} -1 & (s_j[t-1] = 1, s_j[t] = 0), \\ 1 & (s_j[t-1] = 0, s_j[t] = 1). \end{cases} \tag{6}$$

When $t = 0$, the calculation time is equal to that of the conventional Simle-TDC in Eq. (4). However, after the second calculation ($t > 0$), the operation is performed according to Eqs. (5) and (6) only for the changed neurons. Therefore, the calculation time can be significantly reduced compared to that of the conventional Simple-TDC. In [21], Kawashima reported that by introducing S-DMACC to CBMs, the calculation time of a CBM with 300 neurons was approximately one-twentieth that of the conventional Simple-TDC.

III. Proposed LUTNet-RC

In this work, we propose LUTNet-RC, which is LUTNets applied to RC. This work is the first to apply LUTNets to RC. LUTNet-RC consists of a LUT-based reservoir layer and a non-LUT-based output layer.

A. LUT-based reservoir layer

We propose a novel RC model, which is modified from the conventional binarized RC model, binary ESN [23]. The binary ESN is represented by Eqs. (7), (8), (9) and, (10).

$$s_i[t] = sgn(x_i[t]), \tag{7}$$

$$x_i[t] = \sum_{j=1}^{N_r} W_{rr_{i,j}} s_j[t-1] + u[t], \tag{8}$$

$$sgn(x) = \begin{cases} -1 & (x < 0), \\ 1 & (otherwise, \end{cases} \tag{9}$$

$$o_k[t] = b_k + \sum_{j=1}^{N_r} s_j[t] * W_{ro_{k,j}}, \tag{10}$$

where $u[t]$, $s_i[t]$, $x_i[t]$, and $o_k[t]$ are the input value, the external state of ith reservoir neuron, the internal state of ith reservoir neuron, and the output value of kth output neuron at time t, respectively. The reservoir layer of the binary ESN consists of binary neurons $s_i \in \{-1, 1\}$ and binary weights $W_{rr_{i,j}} \in \{-1, 0, 1\}$ (a zero value indicates no connection between ith and jth reservoir neurons).

The binary ESN is controlled by three parameters: (1) k, the number of inputs to one reservoir neuron from other reservoir neurons; (2) p, the asymmetry in $W_{rr_{i,j}}$ values; and (3) N_r, the number of neurons in the reservoir layer. For example, when $k = 100$, one reservoir neuron has inputs from 100 reservoir neurons, and when $p = 1/2$, there is an equal probability of -1 and 1 in W_{rr}. These three parameters control the strength of the chaos of the reservoir layer such that it operates at the edge of chaos, where it performs best [24], [25], [26].

We modified the binary ESN model and propose a novel RC model which is suitable for LUTNets. The proposed RC model is represented by Eqs. (7), (9), (10), and (11).

$$x_i[t] = \sum_{j=1}^{N_r} W'_{rr_{i,j}} * s_j[t-1] + \sum_{j=1}^{N_i} W'_{ir_{i,j}} * u_j[t], \tag{11}$$

where $W'_{rr_{i,j}} \in [-r_{rr}(1-p), r_{rr}p]$ and $W'_{ir_{i,j}} \in [-r_{ir}, r_{ir}]$ are non-binarized weights of the reservoir layer and weights between the input and reservoir layers, respectively, and the connectivity of W'_{ir} is controlled by C_{ir} value. Although weights become multi-bit, the proposed RC model can improve accuracy without increasing the circuit resources because it is implemented using LUTNets. The proposed RC model's strength of chaos is controlled by parameters, k, p, N_r, r_{rr}, and r_{ir}.

B. Non-LUT-based output layer

We propose a novel hardware-oriented algorithm called multi-DMACC (M-DMACC), which is a modification of the conventional S-DMACC for the LUT-based reservoir layer.

The conventional S-DMACC calculates only the changed neurons compared to the previous network state. S-DMACC

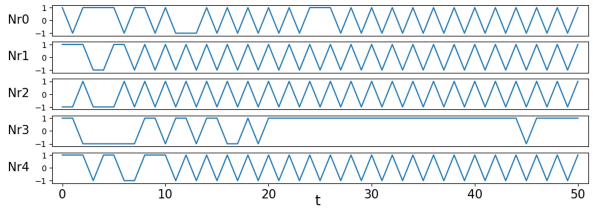


Fig. 7. Temporal change of 5 reservoir neurons in the LUT-based reservoir

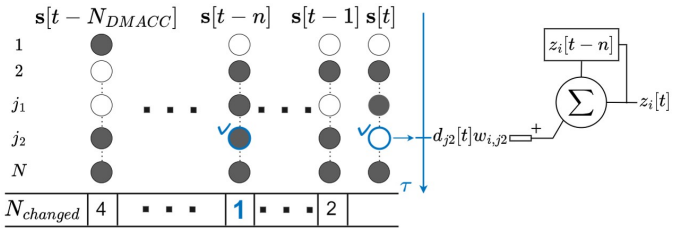


Fig. 8. Proposed time-division calculation method (M-DMACC)

is highly effective and sufficient for models such as CBMs applied in conventional works, where the number of changed neurons is small [21]. On the other hand, we found empirically that the proposed LUT-based reservoir layer has many regularly changing neurons, as shown in Fig. 7, which is the external state $\mathbf{s}$ of five reservoir neurons of the LUT-based reservoir when uniform random values are input. Because of this characteristic of the LUT-based reservoir layer, it would often occur that the number of changed neurons is large when compared to the previous network state, but the number of changed neurons is small when compared to a network state of two times ago. Therefore, we proposed M-DMACC, which performs a DMACC operation starting from the network state with the smallest number of changed neurons $N_{changed}$ by comparing the current network state with before network states, N_{DMACC} times ago, like Fig. 8.

Furthermore, because the external state of the LUT-based reservoir layer is not $\{0, 1\}$ but $\{-1, 1\}$ from Eq. (7), DMACC of the LUTNet-RC is represented by Eqs. (4), (5), and (12). By adding and subtracting twice the value of the weights, DMACC became suitable for LUTNet-RC.

$$d_j[t] = \begin{cases} -2 & (s_j[t-1] = 1,\ s_j[t] = -1), \\ 2 & (s_j[t-1] = -1,\ s_j[t] = 1). \end{cases} \tag{12}$$

IV. Circuit Design of LUTNet-RC

A. LUT-based reservoir layer

The LUT-based reservoir layer is represented by Eqs. (7), (9), (10) and (11). Two types of reservoir neurons exist: those with no input from the input layer and those with input from the input layer. In this work, our target board is AMD's FPGA board; therefore we set $k = 6$ because the FPGA board has six-input, one-output LUT elements (LUT6s). Therefore, the reservoir neurons without input from the input layer were implemented as a six-input, one-output LUT. In addition,

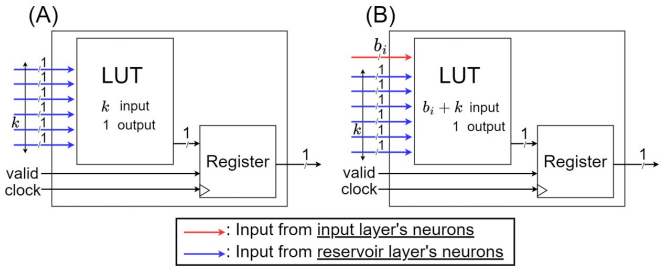


Fig. 9. Circuit design of neurons in the LUT-based reservoir layer

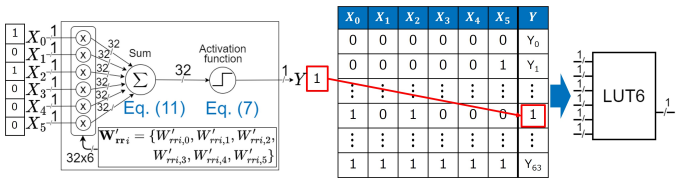


Fig. 10. Mapping a LUT-based reservoir neuron to a LUT element

the reservoir neurons with input from the input layer were implemented as a b_i+6-input, one-output LUT when the bit width of the input data was set to b_i. Because the reservoir layer includes recurrent connections, each reservoir neuron also has a register that is internally controlled by a *valid* signal and the global clock. Reservoir neuron circuits are as shown in Fig. 9(A) and (B) for the reservoir neurons without and with input from the input layer, respectively.

We created a circuit program for the LUT-based reservoir layer in Verilog HDL, which was automatically generated by a software program in Python from the set of hyper-parameters, as shown in Fig. 10. For example, a reservoir neuron without input from the input layer is a six-input, one-output neuron model (left part in Fig. 10). Then, the inputs $\{X_0, X_1, X_2, X_3, X_4, X_5\}$ was $\{1, 0, 1, 0, 0, 0\}$, the output Y was 1 from Eq. (7) and W_{rr}, and this result was filled in the truth table (center part in Fig. 10). In this manner, the output Y is calculated for all 2^6 input patterns, and the output table $\mathbf{Y}$ of the truth table is filled and mapped to a LUT6 (right part in Fig.10). Finally, reservoir neurons are connected, and a circuit program is automatically generated in Verilog HDL using Python programs, as shown in Fig. 11, which includes the register. By performing this for all reservoir neurons, the circuit program becomes equivalent to the reservoir layer defined in Python programs; thus, model validation is easy.

B. Output layer with proposed M-DMACC

The output layer is a multiply-accumulation circuit containing the proposed hardware-oriented algorithm M-DMACC, and we created the circuit program in Verilog HDL. Fig. 12 shows the circuit design of the output neuron with M-DMACC, which compares the current network state with the network states up to the past N_{DMACC} times. The current network state is held in rS_t, the past network states are held in rS_{t-1} to $rS_{t-N_{DMACC}}$, the differences are calculated by performing XORs, and the network state with the smallest difference is

```
1  // LUTrr275
2  wire [64-1:0] lut275_table = 64'b1100110011001100110011001100110011001100110011001100110011001100;
3  wire [6-1:0] lut275_select = {
4                               wData[1495],
5                               wData[258],
6                               wData[1355],
7                               wData[1053],
8                               wData[798],
9                               wData[897]};
10 wire lut275_out = lut275_table[lut275_select];
11 reg lut275_ff;
12 always @(posedge iCLK) begin
13    if (iRST)begin
14       lut275_ff <= 1'b0;
15    end
16    else if (iValid) begin
17       lut275_ff <= lut275_out;
18    end
19 end
20 assign wData[275] = lut275_ff;
```

Output table Y

Connection with other reservoir neurons

Register

Fig. 11. Circuit program of one reservoir neuron in Verilog HDL

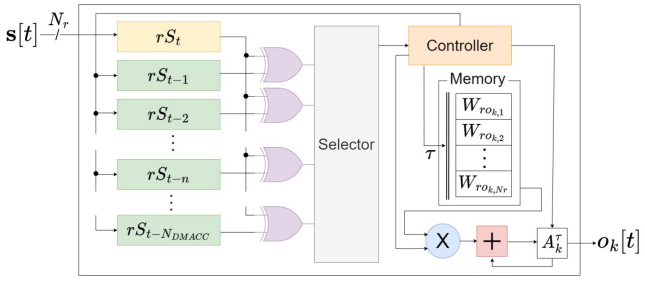


Fig. 12. Circuit design of a neuron in the output layer with M-DMACC

selected using a selector circuit. A controller circuit calculates the address τ to perform DMACC. Using the address τ, W_{ro} stored in memory is input into the multiply-accumulation circuit. After the calculation is complete, the holding network state is updated by adding the current network state $\mathbf{s}[t]$.

When implementing the output layer with S-DMACC, one past network state is stored, and the selector circuit is not required. When implementing the output layer without DMACC, i.e., Simple-TDC, none of the circuits for DMACC described above are required, and the controller circuit increases the address τ by one and calculates all reservoir neurons.

V. Implementations

The implemented system is illustrated in Fig. 13 for an FPGA in the environment listed in Table I. We implemented direct memory access (DMA) cores, a DMApub and DMAsub, to exchange data with the CPU and LUTNet-RC circuit. W_{ro} writes use a block RAM (BRAM) interface consisting of a *valid*, *address*, and *value* signals, whereas other data are exchanged by handshaking using the AXI, AXI-lite, or AXI-stream (AXIS) protocols. We used ridge regression to train W_{ro} in Eq. (13):

$$W_{ro} = \mathbf{d}\mathbf{s}^T(\mathbf{s}\mathbf{s}^T + \lambda\mathbf{I})^{-1}, \tag{13}$$

where $\mathbf{d}$, $\mathbf{s}$ and λ are the vectors of target data $d(t)$, network states $\mathbf{s}[t]$ and the regularization factor, respectively.

We implemented the LUTNet-RC system on an FPGA with hyperparameters listed in Table II, with three different time-division calculations (Simple-TDC, S-DMACC, and M-DMACC) of the output layer, and the utilization of circuit resources are shown in Table III, which running at 100MHz.

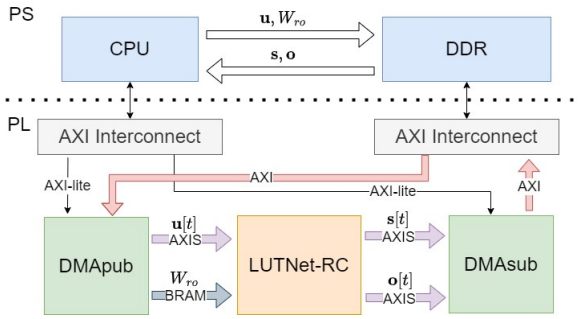


Fig. 13. Implemented LUTNet-RC system on an FPGA

TABLE I
IMPLEMENTATION ENVIRONMENT

Target board	KR260
Device	Zynq UltraScale+ MPSoC EV (XCK26)
Vitis, Vivado, VitisHLS	v2022.2
Python	v3.8.10

In addition, we synthesized only the LUT-based reservoir layer for $b_i = 6, 8, 10, 12$ and obtained all 1490 LUTs and 1470 FFs.

VI. EXPERIMENTS

We evaluated the performance of LUTNet-RC by measuring the memory capacity and nonlinearity and by solving the benchmark task nonlinear auto-regressive moving average 10 (NARMA10) [27]. The number of training data T_{train} and test data T_{test} were 1000 and 500, respectively.

A. Memory capacity and nonlinearity

We conducted experiments with two tasks to investigate the characteristics of LUTNet-RC. The first task was a short-term memory (STM) task used to measure memory capacity [28], which is represented by Eqs. (14) and (15). The STM task involves inferring a uniform random input $S_{in} \in \{0, 1\}$ before the T_{delay} step at time t.

$$S_{in}[t] = 0 \ or \ 1 \ (random), \tag{14}$$

$$y[t]_{T_{delay}} = S_{in}[t - T_{delay}], \tag{15}$$

The second task was a parity-check (PC) task used to measure memory capacity and nonlinearity simultaneously [29] and is represented by Eqs. (14) and (16). The PC task involves inferring whether the sum of inputs before T_{delay} step is odd or even.

$$y[t]_{T_{delay}} = \sum_{i=0}^{T_{delay}} S_{in}[t - i] \ (mod 2), \tag{16}$$

Then, the STM and PC tasks were tested with various T_{delay} values. The coefficient of determination R^2 is represented in Eq. (17) was summed to obtain task scores MC_{STM} and MC_{PC} in Eq. (18). $\hat{y[t]}$ denotes the RC output data at time t.

$$R^2(T_{delay}) = \frac{Cov(y[t]_{T_{delay}}, \hat{y[t]})^2}{Var(y[t]_{T_{delay}}) Var(\hat{y[t]})}, \tag{17}$$

TABLE II
IMPLEMENTED LUTNET-RC'S HYPARPARAMETERS

the bit width of input data $\mathbf{u}$ (b_i)	10
the bit width of W_{ro}	32
$N_i,\ N_r,\ N_o$	1, 1500, 1
k	6
p	0.665
C_{ir}, r_{ir}	0.15, 20
r_{rr}	4
N_{DMACC}	4

TABLE III
UTILIZATION OF CIRCUIT RESOURCES FOR LUTNET-RC SYSTEM

type of the output layer	Utilization LUT (%)	LUTRAM (%)	FF (%)	BRAM (%)
Simple-TDC	6,945 (5.93)	367 (0.64)	12,209 (5.21)	4 (2.78)
S-DMACC	17,901 (15.28)	367 (0.64)	15,247 (6.51)	4 (2.78)
M-DMACC	30,549 (26.08)	387 (0.67)	19,681 (8.40)	4 (2.78)

$$MC_{STM}, MC_{PC} = \sum_{T_{delay}} R^2(T_{delay}). \tag{18}$$

We conducted ten times trials on the two tasks with 1500 reservoir neurons LUTNet-RC, ESN, and CBM-RC with different seeds S_i. The termination conditions of the two tasks were as follows: for the STM task, when T_{delay} was greater than 20 and the average of ten R^2 values was less than 0.01; and for the PC task, when T_{delay} value was greater than 10 and the average of ten R^2 values was less than 0.01.

The results of the STM and PC tasks are presented in Figs. 14 and 15, respectively, and MC_{STM} and MC_{PC} scores are listed in Table IV.

B. NARMA10

We solved NARMA10 [27], represented by Eq. (19), with the FPGA-implemented 1500-reservoir-neuron LUTNet-RC and a software-implemented 1500-reservoir-neuron ESN and CBM-RC. NARMA10 is one of the most well-known benchmark problems for nonlinear time-series tasks.

$$y[t+1] = 0.3y[t] + 0.05y[t]\{\sum_{j=0}^{9} y[t-j]\} + 1.5u[t]u[t-9] + 0.1, \tag{19}$$

where $u[t]$ is a uniform random input between 0 and 0.5. We measured the mean squared error (MSE), the normalized MSE (NMSE), the root MSE (RMSE), and the normalized RMSE (NRMSE), as represented by Eqs. (20), (21), (22), and (23), respectively. $y[t]$ and $\hat{y[t]}$ are target data and RC output data, respectively.

$$MSE = \sum_{i=1}^{T_{test}} (y[i] - \hat{y[i]})^2 / T_{test}, \tag{20}$$

$$NMSE = \frac{\sum_{i=1}^{T_{test}} \{y[i] - \hat{y[i]}\}^2}{\sum_{i=1}^{T_{test}} y[i]^2}, \tag{21}$$

$$RMSE = \sqrt{MSE}, \tag{22}$$

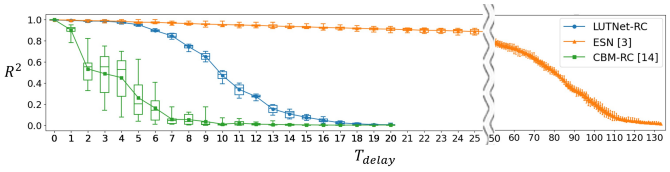


Fig. 14. Experimental results of the STM task

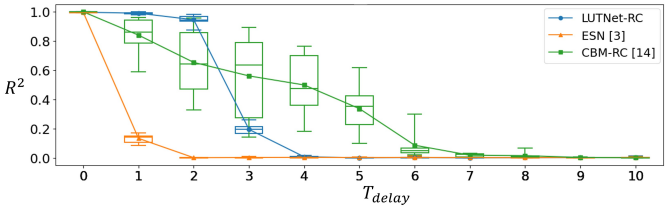


Fig. 15. Experimental results of the PC task

$$NRMSE = \frac{\sqrt{\sum_{i=1}^{T_{test}} \{y[i] - \hat{y[i]}\}^2}}{\sum_{i=1}^{T_{test}} \hat{y[i]}/T_{test}}, \tag{23}$$

The FPGA-implemented LUTNet-RC inference results are shown in Fig. 16, and MSE, $NMSE$, $RMSE$, and $NRMSE$ values are listed in Table IV.

In addition, we estimated the calculation times for three different types of output layers. As a result of the estimation using NARMA10, the output layer with Simple-TDC, S-DMACC, and M-DMACC required 1500 clk, 83 clk, and 60 clk to process one data on average, respectively. Therefore, because all the LUTNet-RC implemented on an FPGA ran at 100MHz, the number of processed samples per second (sps) exceeded more than 10^6 (1Msps) by introducing DMACC.

VII. Discussion

A. LUT-based reservoir layer

The circuit resources of the LUT-based reservoir layer were small, according to the implementation and synthesis results. A method exists for implementing a reservoir with a small circuit resource that implements only one reservoir neuron and uses it repeatedly (virtual-RC), but the circuit resources of LUTNet-RC are comparable to those of virtual-RCs. For example, [35] required two adders, two multipliers, one nonlinear function circuit, and several memory units. A 32-bit precision adder and multiplier require (32 LUTs, 99 FFs) and (717 LUTs, 47 FFs) from the synthesis results, respectively. Therefore, the virtual-RC [35] requires 1498 LUTs, 292 FFs, a nonlinear function circuit, and memory units, which would be larger than the 1490 LUTs and 1470 FFs required by the LUT-based reservoir layer.

Another reservoir with a small circuit resource is the autonomous Boolean network reservoir [38] [39] [40]. However, this method recurrently connects arithmetic elements, such as AND and OR, without the global clock and uses the chaos generated by them as a reservoir, which means that FPGAs are used as analog circuits. This approach makes the model stability and analysis difficult. Furthermore, the method is impractical because the performance depends on the arrangement of elements on the used FPGA board, and the timing of data input depends on the dynamics of the reservoir.

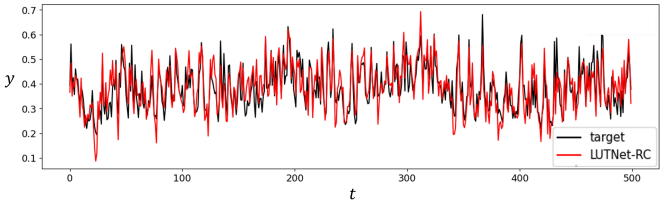


Fig. 16. Experimental result of LUTNet-RC on an FPGA for the NARMA10

TABLE IV
Experimental Results

	LUTNet-RC	ESN	CBM-RC
Nr	1500	1500	1500
MC_{STM}	10.53	**75.20**	4.01
MC_{PC}	3.13	1.15	**4.01**
NARMA10			
MSE	0.00282*	**0.00214**	0.00628
NMSE	0.01769*	**0.01006**	0.03992
RMSE	0.05306*	**0.04626**	0.07925
NRMSE	0.13721*	**0.10150**	0.19344

default is software result, * is FPGA result

Furthermore, from the synthesis results of the LUT-based reservoir layer, the bit width of input b_i did not affect the utilization of circuit resources. In addition, when b_i is 10, a 16-input, one-output LUT is built, which requires $2^{16}/2^6 = 1024$ LUT6s, but the actual synthesis results show that the circuit size was extremely reduced. These were caused by various input weighting and a simple activation function. W_{ir} and W_{rr} were randomly defined values, so each input had a different strength of influence on the output. Some inputs had a relatively minimal and ignorable influence on the output after processing by the binarized activation function in Eq. (9). Therefore, even when a 16-input, one-output LUT was built, it was actually sufficient to refer to only a few bits of the 16-bit input to determine the output. Vivado logic optimization organized such input/output dependencies, causing an extreme reduction in circuit size. In Fig. 11, the output depended only on the input $wData[798]$.

In terms of performance, as shown in Figs. 14 and 15 and Table IV, both the memory capacity and nonlinearity of LUTNet-RC were high, and the results of NARMA10 were comparable to those of ESN. The importance of nonlinearity in RC has been highlighted [12], and LUTNet-RC will have advantages in tasks that require both memory capacity and nonlinearity.

B. Output layer with DMACC

The experimental results show that by introducing DMACC, the calculation time was significantly reduced, and the inference was performed faster than 1Msps. Furthermore, the output layer with the proposed M-DMACC was approximately 1.4 times as fast as the output layer with the conventional S-DMACC. Although DMACC is a highly powerful hardware-

TABLE V
FPGA IMPLEMENTATIONS OF RC

	RC type	Accuracy		Speed		Circuit resources					
		N_r	NARMA10	N_r	throughput	module	N_r	Utilization			
								LUT	FF	DSP	BRAM
LUTNet-RC Simple-TDC	ESN	1500	**MSE=0.00282 NMSE=0.01769 RMSE=0.05306 NRMSE=0.13721**	1500	0.067Msps	RC	**1500**	6945	12209	0	4
S-DMACC					**1.2Msps**			17901	15247	0	4
M-DMACC					**1.7Msps**			30549	19681	0	4
[30] (2020)		100	(NMSE=0.1250)	32	**2.6Msps**		16†	10967	7203	162	12
[31] (2022)		-	-	16	**1.2Msps**		16†	2133	5978	16	0
[32] (2020)		600	MSE=0.0030*	100	0.20Msps		100	28933	44021	20	48
[33] (2020)		-	-	300	0.035Msps		300	4071	5497	6	12.5
[34] (2016)		-	-	50	25sps		50	5306 Logic elements			
[35] (2021)	ESN (virtual)	400	NRMSE=0.159	400	0.12Msps	RC + trainer	400	(CycloneIV EP4CE10F17C8)			
[36] (2018)		-	-	18	0.31Msps		18	(Virtex6: 7% of LUTs and 77% of BRAMs)			
[37] (2008)	LSM (virtual)	-	-	200	0.024Msps	RC	10	10118	-	103	30
[22] (2021)	CBM-RC	1500	NMSE=0.040*	1500	0.033Msps		2048†	1089461	593285	2048	1 + 256††
[26] (2021)	PCPO-RC	100	RMSE=0.249	100	0.23Msps	reservoir	100	2283	1695	0	0

Accuracy: default is FPGA result, * is software result. () value is uncertain of definition.
Circuit resources: † means synthesis result without interface and †† is the number of ultra RAM, which are larger RAM elements than BRAMs on FPGAs.

oriented algorithm that can reduce the increase in calculation time caused by time-division calculations without decreasing accuracy, it causes an increase in circuit resources. However, some applications require higher calculation speed at the expense of increased circuit resources. Therefore, the introduction of DMACC in the output layer is important for LUTNet-RC because, unlike other small circuit resource reservoirs, the LUT-based reservoir layer completes its calculation in 1 clk; therefore, the overall calculation speed of LUTNet-RC is limited by the calculation speed of the output layer. Furthermore, the trade-off between circuit resources and calculation time can be controlled by increasing or decreasing N_{DMACC} and whether DMACC is introduced or not. These results and analyses show that LUTNet-RC is practical and allows for the implementation of RC systems that are fitted to applications.

C. Comparison with conventional works

Table V shows a comparison of LUTNet-RC with conventional RCs on FPGAs in terms of calculation accuracy, speed, and circuit resources. In terms of accuracy, the LUTNet-RC had the best score for RC on FPGAs when solving NARMA10. However, because an indice is an uncertain definition in [30] and other RC systems on FPGAs have been verified in various tasks such as speech recognition, it is necessary to verify the LUTNet-RC in other tasks as well. In terms of speed, the LUTNet-RC is one of the few RCs on FPGAs that exceed 1Msps. The throughput reported in [30] was 2.6Msps, but N_r was small and the accuracy in this situation was low; if N_r is increased to 100 or 1500 to obtain higher accuracy, it causes a serious increase in calculation time, making the throughput smaller than 1Msps. In terms of circuit resources, the LUTNet-RC has a large number of reservoir neurons N_r with small circuit resources. Since the performance of RC improves with an increase in the number of reservoir neurons [3], the implementation of RC using the proposed method is highly powerful.

When considering the three indices together, the LUTNet-RC achieves a high level of coexistence; therefore, we conclude that the LUTNet-RC proposed and implemented in this work is one of the highest-performing RC on an FPGA.

VIII. CONCLUSION

In this work, we proposed LUTNet-RC, which consists of a LUT-based reservoir layer and a non-LUT-based output layer. In the LUT-based reservoir layer, we modeled a multi-bit weight reservoir, modifying the conventional binarized reservoir to improve calculation accuracy. In the non-LUT-based output layer, we introduced DMACC to reduce the increase in the calculation time caused by time-division calculation. Furthermore, because conventional S-DMACC is not suited for the LUT-based reservoir, which has many regular node changes, we proposed and introduced M-DMACC, which compares the network state several times ago and calculates only the changed neurons.

We implemented LUTNet-RC with 1500 reservoir neurons on an FPGA running at 100MHz. We measured the memory capacity and nonlinearity of LUTNet-RC and solved NARMA10 using the LUTNet-RC. These scores were comparable to those of conventional RC on FPGAs, and the LUTNet-RC can process more than 10^6 data per second. We conclude that the LUTNet-RC proposed and implemented in this work is one of the highest-performance RC on an FPGA. In the future, we plan to implement an RC system that can run on an FPGA alone by combining LUTNet-RC and a learning accelerator [41].

ACKNOWLEDGEMENT

This work is based on results obtained from a project, JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and JSPS KAKENHI Grant Number 23H03468. We gratefully acknowledge support from TIER IV Inc.

REFERENCES

[1] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[2] M. Groshev, G. Baldoni, L. Cominardi, A. de la Oliva, and R. Gazda, "Edge robotics: Are we ready? An experimental evaluation of current vision and future directions," *Digital Communications and Networks*, vol. 9, no. 1, pp. 166–174, 2023.

[3] 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, no. 34, p. 13, 2001.

[4] C. Gallicchio, "Euler State Networks: Non-dissipative Reservoir Computing," *arXiv preprint arXiv:2203.09382*, 2022.

[5] J. Moon, Y. Wu, and W. D. Lu, "Hierarchical architectures in reservoir computing systems," *Neuromorphic Computing and Engineering*, vol. 1, no. 1, p. 014006, 2021.

[6] F. Nowshin, Y. Zhang, L. Liu, and Y. Yi, "Recent advances in reservoir computing with a focus on electronic reservoirs," in *2020 11th International Green and Sustainable Computing Workshops (IGSC)*. IEEE, 2020, pp. 1–8.

[7] E. Wang, J. J. Davis, P. Y. Cheung, and G. A. Constantinides, "LUTNet: Rethinking inference in FPGA soft logic," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 26–34.

[8] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[9] R. Fuchikami and F. Issiki, "Fast and light-weight binarized neural network implemented in an fpga using lut-based signal processing and its time-domain extension for multi-bit processing," in *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*. IEEE, 2019, pp. 120–121.

[10] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[11] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, "Memory versus non-linearity in reservoirs," in *The 2010 international joint conference on neural networks (IJCNN)*. IEEE, 2010, pp. 1–8.

[12] M. Inubushi and K. Yoshimura, "Reservoir computing beyond memory-nonlinearity trade-off," *Scientific reports*, vol. 7, no. 1, p. 10199, 2017.

[13] W. Maass, T. Natschläger, 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, 2002.

[14] Y. Katori, H. Tamukoh, and T. Morie, "Reservoir computing based on dynamics of pseudo-billiard system in hypercube," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[15] D. Pramanta and H. Tamukoh, "Design and implementation of pulse-coupled phase oscillators on a field-programmable gate array for reservoir computing," in *International Conference on Neural Information Processing*. Springer, 2020, pp. 333–341.

[16] Y. Horio, "Chaotic neural network reservoir," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–5.

[17] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "LogicNets: Co-designed neural networks and circuits for extreme-throughput applications," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 291–297.

[18] N. Soga and H. Nakahara, "Design Method for an LUT Network-Based CNN with a Sparse Local Convolution," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 294–295.

[19] S. Chatterjee, "Learning and memorization," in *International conference on machine learning*. PMLR, 2018, pp. 755–763.

[20] C. Kiefer, "Stochastic Optimisation of Lookup Table Networks, for Realtime Inference on Embedded Systems," in *Proceedings of the 2nd Joint Conference on AI Music Creativity*, p. 10.

[21] I. Kawashima, T. Morie, and H. Tamukoh, "FPGA implementation of hardware-oriented chaotic Boltzmann machines," *IEEE Access*, vol. 8, pp. 204 360–204 377, 2020.

[22] I. Kawashima, Y. Katori, T. Morie, and H. Tamukoh, "An area-efficient multiply-accumulation architecture and implementations for time-domain neural processing," in *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2021, pp. 1–4.

[23] P. Verzelli, L. Livi, and C. Alippi, "A characterization of the edge of criticality in binary echo state networks," in *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2018, pp. 1–6.

[24] P. Barančok and I. Farkaš, "Memory capacity of input-driven echo state networks at the edge of chaos," in *Artificial Neural Networks and Machine Learning–ICANN 2014: 24th International Conference on Artificial Neural Networks, Hamburg, Germany, September 15-19, 2014. Proceedings 24*. Springer, 2014, pp. 41–48.

[25] L. Livi, F. M. Bianchi, and C. Alippi, "Determination of the edge of criticality in echo state networks through Fisher information maximization," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 3, pp. 706–717, 2017.

[26] D. Pramanta and H. Tamukoh, "FPGA Implementation of Pulse-Coupled Phase Oscillators working as a Reservoir at the Edge of Chaos," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[27] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: unifying the algorithms and accelerating convergence," *IEEE transactions on neural networks*, vol. 11, no. 3, pp. 697–709, 2000.

[28] H. Jaeger, "Short-term memory in echo states networks," *Technical Report GMD Report 152*, 2002.

[29] N. Bertschinger and T. Natschläger, "Real-time computation at the edge of chaos in recurrent neural networks," *Neural computation*, vol. 16, no. 7, pp. 1413–1436, 2004.

[30] 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.

[31] 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.

[32] K. Honda and H. Tamukoh, "A hardware-oriented echo state network and its FPGA implementation," *Journal of Robotics, Networking and Artificial Life*, vol. 7, no. 1, pp. 58–62, 2020.

[33] D. Kleyko, E. P. Frady, M. Kheffache, and E. Osipov, "Integer echo state networks: Efficient reservoir computing for digital hardware," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1688–1701, 2020.

[34] M. L. Alomar, V. Canals, N. Perez-Mora, V. Martínez-Moll, and J. L. Rosselló, "FPGA-based stochastic echo state networks for time-series forecasting," *Computational intelligence and neuroscience*, vol. 2016, pp. 15–15, 2016.

[35] K. Yoshida, Y. Abe, M. Akai-Kasaya, and T. Asai, "FPGA Architecture for Reservoir Computing with time-division input interfaces and online learning systems," *IEICE Technical Report; IEICE Tech. Rep.*, 2021.

[36] B. Penkovsky, L. Larger, and D. Brunner, "Efficient design of hardware-enabled reservoir computing in FPGAs," *Journal of Applied Physics*, vol. 124, no. 16, 2018.

[37] B. Schrauwen, M. D'Haene, D. Verstraeten, and J. Van Campenhout, "Compact hardware liquid state machines on FPGA for real-time speech recognition," *Neural networks*, vol. 21, no. 2-3, pp. 511–523, 2008.

[38] D. Canaday, A. Griffith, and D. J. Gauthier, "Rapid time series prediction with a hardware-based reservoir computer," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 12, 2018.

[39] H. Komkov, L. Pocher, A. Restelli, B. Hunt, and D. Lathrop, "RF signal classification using Boolean reservoir computing on an FPGA," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–9.

[40] S. Apostel, N. D. Haynes, E. Schöll, O. D'Huys, and D. J. Gauthier, "Reservoir Computing Using Autonomous Boolean Networks Realized on Field-Programmable Gate Arrays," *Reservoir Computing: Theory, Physical Implementations, and Applications*, pp. 239–271, 2021.

[41] K. Yoshida, M. Akai-Kasaya, and T. Asai, "A 1-Msps 500-Node FORCE Learning Accelerator for Reservoir Computing," *Journal of Signal Processing*, vol. 26, no. 4, pp. 103–106, 2022.