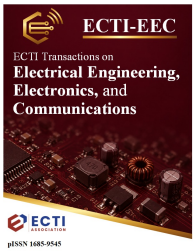
ECTI-EEC Transactions Pre-proof




# Delayed Log-Domain Reservoir Computing for Ultra-Low-Resource FPGAs

Patinya Ketthong, Nattagit Jiteurtragool, and Chatchai Wannaboon

DOI: http://doi.org/10.37936/ecti-eec.2027251.266979

Reference: ECTI-EEC 266979

Received date: 21 July 2026

Accepted date: 7 October 2026

This is a PDF of an article that has been improved after acceptance, including the inclusion of a cover page and metadata, as well as the optimization of readability. Before its publication in its final form, this version will undergo further copyediting, typesetting, and review. As a result, this version is no longer the Accepted Manuscript, but it is not yet the definitive **Version of Record**. We are providing this early version to provide the article with early visibility.

**Published by ECTI Association Thailand**

ECTI Transactions on Electrical Engineering, Electronics, and Communications, Vol. 25, No. 1, pp. 1–10, February 2027

https://doi.org/10.37936/ecti-eec.2027251.266979
ISSN(Online): 1685-9545

# Delayed Log-Domain Reservoir Computing for Ultra-Low-Resource FPGAs

Patinya Ketthong[1], Nattagit Jiteurtragool[2] and Chatchai Wannaboon[3†]

## Abstract

Reservoir computing provides a low-training-cost alternative to recurrent neural networks for temporal tasks, but large arithmetic requirements limit its deployment on ultra-low-resource field-programmable gate arrays (FPGAs) lacking dedicated multiplier blocks. We present a streaming reservoir computing architecture that heavily reduces look-up table (LUT) utilization by computing the recurrent state update in the log domain using an approximate Mitchell multiplier. By restricting computationally intensive exact multiplication to the readout layer—which isolated synthesis confirms is the primary soft-logic bottleneck—and unifying the delay buffer, recurrent state, and readout weights into a single time-multiplexed block RAM, the architecture minimizes logic requirements while maintaining temporal memory capacity. Evaluated on the non-linear autoregressive moving average (NARMA-10) benchmark, the proposed design achieves a normalized root mean square error (NRMSE) of $0.3544 \pm 0.0359$, showing no statistically resolved accuracy degradation compared to an exact baseline. Implemented on a Lattice iCE40 HX8K FPGA without dedicated DSP blocks, the design reduces LUT4 utilization by 19.85% and placed logic cells by 18.65% relative to an exact LUT-based baseline, demonstrating a highly efficient inference solution exclusively tailored for area-constrained edge fabrics lacking dedicated multiplier hardware.

**Keywords:** Approximate arithmetic, Delayed feedback, Multiplier-less FPGA, Logarithmic number system, NARMA-10, Reservoir computing, Streaming architecture.

## I. Introduction

Recurrent neural networks can model temporal signals, but conventional training requires gradients through many time steps and retains many trained recurrent parameters. Reservoir computing (RC) instead fixes the nonlinear recurrent system and fits only a linear output layer [1, 2]. Carefully chosen nonlinear feature maps can further simplify the recurrent construction while retaining forecasting performance [3]. The resulting ridge-regression training problem is attractive when adaptation occurs off chip but inference must fit an edge device.

A conventional digital echo-state network (ESN) maintains recurrent states, applies a recurrent matrix and nonlinear activation, and forms a weighted output. Dense products scale quadratically with reservoir size. Quantization and structured recurrence reduce arithmetic, but not necessarily state and readout storage. Counting only the nonlinear update can therefore understate the cost of a complete inference engine.

Time multiplexing reuses one nonlinear path to create virtual nodes coupled through a delay. Single-node delayed reservoirs were established experimentally [4], and recent reviews document their extension across physical substrates [5, 6]. In an FPGA, the same concept replaces a spatial array with one recurrent datapath and addressable delay, trading latency for less parallel arithmetic. Without delayed state, however, sequential evaluations remain repeated scalar updates rather than a high-dimensional temporal representation. In this work, *ultra-low-resource* denotes FPGA deployment without usable dedicated multiplier blocks and under tight LUT and block-memory budgets; it does not imply that approximate arithmetic is advantageous on every FPGA family.

Approximate logarithmic multiplication is complementary to time multiplexing. Mitchell multiplication approximates a product through leading-one detection, exponent addition, and piecewise reconstruction instead of a full multiplier [7]. Recent multiplier studies show that the best accuracy–cost point depends on synthesis target and application sensitivity [8, 9]. Approximate-computing arguments must therefore be evaluated at task level because local numerical error can be attenuated or amplified by recurrent dynamics. Arithmetic cell

Manuscript received July 21, 2026; revised on August 24, 2026; accepted October 7, 2026.
†Corresponding Author: chatchai.w@cit.kmutnb.ac.th
[1]Intelligent Electronics Systems Laboratory (IES), Faculty of Engineering, Thai-Nichi Institute of Technology, Bangkok, Thailand.
[2]Department of Computer and Information Science, Faculty of Applied Science, King Mongkut's University of Technology North Bangkok, Thailand.
[3]Department of Electronics Engineering Technology, College of Industrial Technology, King Mongkut's University of Technology North Bangkok, Thailand.

counts alone cannot establish that the complete learned system remains useful.

Existing FPGA reservoirs have demonstrated structured recurrence, delayed feedback, and reduced-cost arithmetic, but these ideas are usually evaluated separately or within systems optimized for different training and deployment objectives [10-13]. It therefore remains unclear whether an inference-only delayed reservoir can combine unified state and weight storage, streaming feature consumption, and approximate recurrent multiplication while preserving a task-level advantage under a controlled exact-arithmetic comparison.

This paper addresses that gap by asking whether a delayed streaming reservoir using signed approximate log-domain products can retain a measurable task advantage over inexpensive baselines while reducing LUT logic relative to a matched exact fixed-point implementation on a multiplier-less FPGA (i.e., FPGAs that do not contain dedicated DSP blocks). The resulting 100-virtual-node inference architecture includes a 128-word recurrent delay, deterministic input masks, piecewise-linear activation, a trained 16-bit readout, output bias, defined reset behavior, and a transaction-level interface. Delay samples and readout weights occupy a unified 256-by-16 memory map, and each feature contributes to the output as soon as it is generated. This streaming organization removes the need for a separate 100-word feature buffer.

The evaluation separates model selection from confirmation. A development-grid study is followed by five independent confirmation sequences, a readout-width ablation, stand-alone arithmetic-error characterization, and comparisons with predictive baselines. The Python reference and RTL are checked for exact agreement, while approximate- and exact-multiplier variants are synthesized and routed with the same device, schedule, memory organization, timing constraint, and placement seed. Together, these measures connect numerical behavior and predictive accuracy to a controlled hardware-cost comparison.

The remainder of the paper first positions the design against digital and delayed RC systems. It then derives the reservoir update and fixed-point arithmetic, describes the streaming microarchitecture, gives the experimental protocol, presents task and FPGA results, and discusses the limits of the obtained operating point.

## II. BACKGROUND AND RELATED WORK

### A. Reservoir Dynamics and Structured Recurrence

A leaky ESN with input $u_t$ and state $X_t$ commonly uses.

$$X_t = (1-\alpha)X_{t-1} + \alpha f(W_{\text{in}}u_t + W_{\text{res}}X_{t-1}) \qquad (1)$$

where $\alpha$ is the leak rate and f is a saturating nonlinearity. A linear readout $\hat{y}_t = b + W^T X_t$ is fitted while $W_{in}$ and $W_{res}$ remain fixed. The useful regime balances fading memory with nonlinear separation. If the dynamics forget too quickly, long dependencies are lost; if recurrence is unstable, small perturbations dominate the input-driven trajectory.

Dense random recurrence is not essential. Integer states and cyclic shifts can avoid general recurrent matrix multiplication while retaining predictive capability [14]. Recent FPGA designs further reduce storage and arithmetic through sparsity-constrained recurrence [12], or through polynomial activation and regenerated internal weights [13]. Application-oriented systems have also integrated the reservoir and readout for signal processing such as ECG denoising [15]. These results show that RC should be co-designed at the dynamical, numerical, and architectural levels rather than obtained by directly mapping a dense software network.

### B. Delay-Based Virtual Nodes

Appeltant *et al.* showed that one nonlinear delayed-feedback system can emulate virtual nodes sampled within an input interval [4]. Masks and timing offsets diversify these states, and current implementations span photonic, mechanical, magnetic, memristive, and electronic systems [5, 6].

The same organizing idea applies to a synchronous digital design, although the physical interpretation differs. Virtual nodes are sequential state updates executed by one arithmetic path. A circular buffer supplies an older state at each update, so the delay creates coupling beyond the current scalar register. FPGA demonstrations include a chain-structured time-delay reservoir [10]and a delayed-feedback edge system with online training and inference [11]. The present design focuses instead on inference-only resource reduction, deterministic bit-level reproduction, and a matched comparison of approximate and exact recurrent arithmetic.

### C. Logarithmic and Approximate Arithmetic

In a logarithmic number system, multiplication becomes addition of logarithms. Mitchell's method approximates the logarithm and antilogarithm using the position of the leading one and a linear fractional term [7]. Current designs add pipelining, correction, or run-time accuracy modes to trade numerical error against throughput and energy [9, 16]. A recent comparative survey shows that these tradeoffs must be evaluated under consistent synthesis and application conditions [8].

The appropriate point depends on the surrounding algorithm. This work does not convert the complete reservoir into a stored logarithmic representation. States and weights remain signed fixed point, and only selected products use a Mitchell-style combinational operator. This avoids logarithmic addition, which is substantially more complex than logarithmic multiplication, and keeps the memory interface conventional. The exact baseline retains the same signed formats and replaces those approximate products with ordinary fixed-point multiplication. As a result, the hardware comparison measures the incremental effect of multiplier choice within the same complete LUT-mapped system.

## III. RESERVOIR MODEL AND DESIGN RATIONALE

### A. From Scalar Recurrence to Delayed State

The design processes one scalar input $u_t$ through $N$ virtual-node updates. Let $i \in \{0, \ldots, N-1\}$ denote the node position within input time $t$. A circular memory of depth $D$ stores previous generated states. Its read value at node $i$ is denoted

$d_{t,i}$. The write pointer advances after every node and persists across transactions, so the address sequence is continuous over input samples.

The internal update is

TABLE I
LOCKED DEPLOYMENT PARAMETERS

| Parameters | Value |
|---|---|
| Virtual nodes $N$ | 100 |
| Circular state delay $D$ | 128 words |
| State and input format | signed Q1.15 |
| Feedback gain $\rho$ | 1.05 (34,406 in Q1.15 scale) |
| Input scale | $2^{-3}$ |
| Leak rate $\alpha$ | 1/2 |
| Mask generator | 16-bit Galois LFSR |
| Activation | 16 knots with interpolation |
| Readout weights | signed 16-bit, scale $2^{-7}$ |
| Readout bias | signed 19 bit |
| Ridge coefficient | $10^{-8}$ |

$$p_{t,j} = \mathcal{M}(d_{t,j}, \rho) + 2^{-3}\mathcal{M}(m_j, u_t) \tag{2}$$

$$x_{t,j} = \left\lfloor \frac{x_{t,j-1} + G(p_{t,j})}{2} \right\rfloor \tag{3}$$

where $\rho$ is the feedback gain, $m_i$ is a signed input mask, $G$ is the activation, and $\mathcal{M}$ is either approximate or exact signed Q1.15 multiplication. The factor $2^{-3}$ is applied after product quantization as an arithmetic right shift. The scalar $x_{t,i-1}$ occupies a register, with $x_{t,-1} = x_{t-1,N-1}$. After (3), $x_{t,i}$ overwrites the addressed delay word and becomes the state used by the next node. The delay read and scalar recurrence therefore provide two distinct memory paths: the immediately preceding state and the state last written $D$ node intervals earlier.

For the selected $N = 100$ and $D = 128$, one input advances the circular buffer by 100 positions rather than a complete revolution. Consequently, the delayed state observed at a given virtual-node position changes its alignment across successive inputs. This controlled mismatch increases mixing without requiring a programmable interconnection matrix. A zero-delay scalar design cannot reproduce this behavior because it has no independent delayed trajectory.

### B. Deterministic Input Mask

A 16-bit maximal-length Galois linear-feedback shift register (LFSR) generates the masks. It is initialized to hexadecimal ACE1 at the beginning of every input transaction. If $r_i$ is the unsigned LFSR word, the signed Q1.15 mask is

$$m_i = \text{signed}(r_i - 2^{15}) \tag{4}$$

The sequence has period 65,535 and the first 100 values are used. Restarting the LFSR makes the mask deterministic without storing 100 words. The delay pointer is not restarted: the mask defines node identity, whereas circular memory carries temporal context.

### C. Hyperparameters and Numerical Formats

Table I summarizes the locked configuration. Feedback $\rho$=1.05 is represented by the 17-bit integer 34,406, allowing a value slightly above the positive Q1.15 state range. Input scaling $2^{-3}$ is implemented as an arithmetic shift. Leakage $\alpha = 1/2$ requires only signed addition and a shift. These power-of-two choices reduce control and coefficient storage, but they were retained only after development-sequence evaluation.

## IV. FIXED-POINT ARITHMETIC

### A. Signed Mitchell Product

For a nonzero unsigned magnitude a, write

$$a = 2^{k_a}(1 + f_a), \qquad 0 \le f_a < 1 \tag{5}$$

where $k_a = \lfloor \log_2 a \rfloor$ Mitchell's approximation uses

$$\log_2 a \approx k_a + f_a \tag{6}$$

For two magnitudes, exponents and fractional fields are added. With $s = f_a + f_b$, reconstruction is

$$\widetilde{ab} = \begin{cases} 2^{k_a+k_b}(1+s), & s < 1 \\ 2^{k_a+k_b}s, & s \ge 1 \end{cases} \tag{7}$$

The RTL retains 12 normalized fractional bits. Leading-one indices come from priority encoders, the two fractions share an adder, and the reconstructed magnitude is shifted back to the Q1.15 scale. Operand signs are XORed and applied after magnitude reconstruction. A zero operand bypasses the leading-one path.

This product is used twice per node in (2): once for delayed feedback and once for masked input injection. The same combinational function is time-shared across the corresponding controller phases. The readout product is not approximated. This choice confines the approximation to state generation and prevents 100 product errors from accumulating directly in the output sum.

The exact configuration changes one RTL parameter: it evaluates a signed 17-by-17-bit product followed by an arithmetic 15-bit shift. Activation, leakage, memory ordering, readout precision, and saturation remain identical, isolating the multiplier choice.

### B. Interpolated Activation

An exact hyperbolic tangent is too costly for the intended device. The selected activation stores 16 Q1.15 samples of $\tanh(z)$ uniformly over $0 \le z \le 2$. For magnitude $q = |p|$, the interval index and residual are

$$j = \left\lfloor \frac{15|a|}{2} \right\rfloor \tag{8}$$

$$r = \frac{15|a|}{2} - j \tag{9}$$

noindent For $j < 15$, linear interpolation gives

$$g(a) = h_j + r(h_{j+1} - h_j) \tag{10}$$

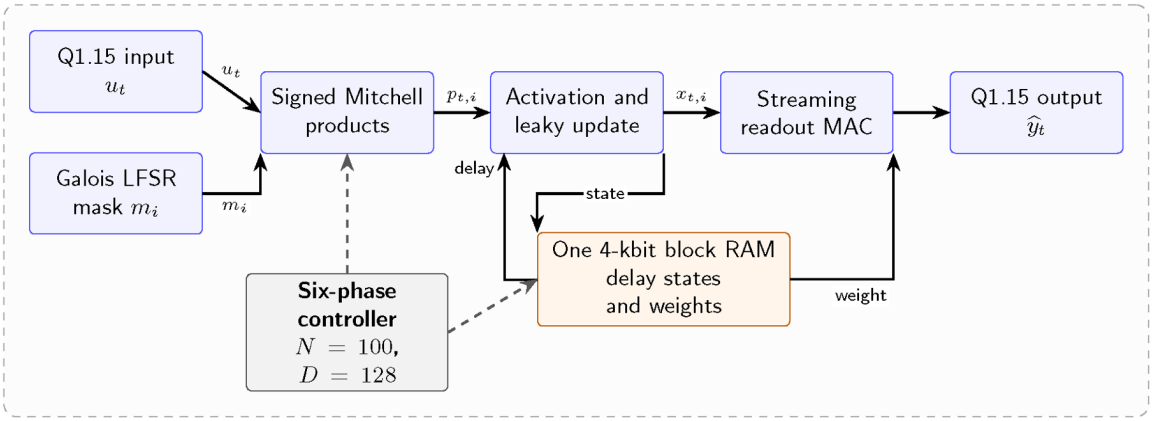



Fig. 1. Delayed Streaming Inference Architecture with Unified State and Weight Memory

where $h_j$ is the $j$th quantized knot. Between magnitudes 2 and 3, a linear tail connects the final knot to the Q1.15 saturation level. Magnitudes at or above 3 saturate, and odd symmetry supplies negative outputs.

The interpolation product uses the general arithmetic inferred by synthesis rather than the Mitchell operator because its operands are an unsigned residual and a small positive knot difference. Separating this path simplified bit-exact agreement with the reference model. It also explains why removing recurrent exact multipliers does not eliminate every multiplication structure from the netlist.

### C. Rounding, Saturation, and Accumulator Width

All right shifts are arithmetic for signed values. Leakage uses floor behavior inherited from the signed shift; this matters for negative odd sums. The state update remains within 16 bits because both the old state and activation lie in the Q1.15 range and their average cannot exceed that interval. Intermediate recurrent and injected values use 19 or 20 bits before activation.

The readout accumulator is 48 bits. A worst-case sum of 100 signed 16-by-16-bit products requires fewer than 39 magnitude bits, so the chosen width provides guard bits beyond the theoretical requirement. After summation, the accumulator is shifted by the learned weight fraction count, the 19-bit bias is added, and the result is saturated to signed Q1.15. Saturation occurs only at the external output; state and product paths use their explicitly sized integer representations.

## V. STREAMING FPGA ARCHITECTURE

### A. Datapath and Unified Memory

Fig. 1 shows the complete data path. The design accepts one signed Q1.15 sample through a valid/ready interface. The input is latched for the duration of 100 virtual-node updates. A shared arithmetic path reads a delayed state, forms the recurrent and injected terms, evaluates the activation and leakage, writes the updated state, reads the corresponding trained weight, and accumulates the readout product.

The inferred 256-by-16 memory has the map in Table II . Addresses 0–127 implement the circular delay. Addresses

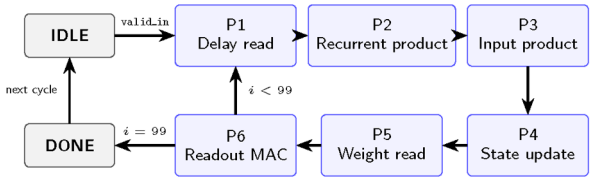



Fig. 2. Six-State Controller Loop for Streaming Virtual-Node Evaluation.

128–227 store 100 trained weights. The remaining 28 words are unused and permit the complete structure to fit one 4-kbit iCE40 block RAM. The 16 activation knots are elaboration-time constants and synthesize into logic. The 19-bit output bias is a parameter constant.

Streaming accumulation is the key distinction between recurrent state and readout state. The 100 $x_{t,i}$ values are features for the current input, but they need not be retained as a second vector. Each value is multiplied by $w_i$ immediately after generation and then discarded from the readout path. Only the copy written into the circular delay persists for future recurrence. A conventional two-stage schedule would require another 1,600-bit feature array or a second pass through state memory.

### B. Controller Schedule and Latency

The controller uses the six-state node loop shown in Fig. 2. The synchronous memory introduces registered read behavior, so address presentation and data consumption occupy separate phases. The approximate operator is used during the delayed-product and input-product states. The activation is evaluated after both terms have been registered. State P4 also writes $x_{t,i}$ into the delay and requests weight address $128 + i$; state P6 consumes that weight and accumulates $w_i x_{t,i}$.

After node 99, one completion interval scales the accumulator, adds the bias, saturates the output, and asserts valid_out. Request-to-result latency is therefore

$$L = 6N + 1 = 601 \quad cycles \tag{11}$$

The architecture is not internally overlapped across inputs; ready remains low until completion. At the routed approx-

TABLE II
ON-CHIP STORAGE ALLOCATION

| Content | Address | Words | Bits |
|---|---|---|---|
| Delay states | 0–127 | 128 | 2,048 |
| Readout weights | 128–227 | 100 | 1,600 |
| Unused memory | 228–255 | 28 | 448 |
| Activation LUT | logic | 16 | 256 |
| Readout bias | logic | 1 | 19 |

imate frequency, the maximum sustained rate is consequently about 39.2 thousand inputs per second. Applications needing a higher sample rate could replicate the core or pipeline controller phases, but either change would alter the reported area point.

### C. Reset, Persistence, and Deployment Parameters

Reset clears the 128 delay words sequentially, initializes the state register and delay pointer, and restores the LFSR seed. Weight addresses are not scrubbed, so the memory image loaded at configuration remains intact. The reset cost is 128 cycles and occurs only at initialization. During normal operation, the circular pointer persists and the LFSR restarts per input, implementing the state/mask distinction described earlier.

Training is performed offline. The deployed hardware contains the resulting quantized weights, bias, and fixed shift. It does not contain a matrix solver or target memory. This inference-only boundary is appropriate for static edge models but differs from systems that provide online readout adaptation. The provided generation script converts trained parameters into both a 100-word weight file and the combined 256-word memory image used by RTL simulation and synthesis.

## VI. EXPERIMENTAL METHODOLOGY

### A. NARMA-10 Task and Data Separation

NARMA-10 is a nonlinear autoregressive moving-average benchmark that requires both temporal memory and nonlinear processing [17]. Inputs are independently sampled from [0,0.5], and the target is

$$y_{t+1} = 0.3y_t + 0.05y_t \sum_{j=0}^{g} y_{t-j} + 1.5u_{t-9}u_t + 0.1 \qquad (12)$$

Each sequence contains 5,200 samples. The first 200 are discarded as washout, followed by 2,800 training, 800 validation, and 1,000 test samples. One fixed pseudorandom sequence is used for development. Delay depth, node count, recurrent gain, input scale, leak, activation form, ridge coefficient, and candidate formats are examined without using its test segment.

After the architecture, reservoir hyperparameters, and training procedure are frozen, five independent confirmation sequences, denoted C1–C5, are generated with fixed pseudorandom seeds. For reproducibility, the development seed is 20260721 and the C1–C5 seeds are 20260722–20260726, respectively. For each confirmation sequence, a separate readout is fitted using the 2,800 training and 800 validation samples, and evaluation uses only its 1,000-sample test segment. Confirmation therefore measures input-sequence variability under a fixed model-selection protocol rather than reusing one readout across independently generated supervised data sets.

The main metric is normalized root-mean-square error

$$\mathrm{NRMSE} = \frac{\sqrt{T^{-1}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}}{\mathrm{std}(y)} \qquad (13)$$

Reported dispersion is the sample standard deviation across the five confirmation seeds. Paired 95% confidence intervals are computed from per-seed NRMSE differences using Student's $t$ distribution with four degrees of freedom. With only five sequences, these intervals are descriptive uncertainty summaries rather than broad population guarantees.

### B. Readout Training and Quantization

Reservoir features are converted from Q1.15 to floating point for offline fitting. Each feature is standardized using fitting-set mean and standard deviation. Ridge regression includes an unpenalized intercept and coefficient $10^{-8}$ on feature weights. Standardized coefficients are then transformed back to the original feature scale.

For B-bit signed readout weights, the largest power-of-two fractional scale that keeps every coefficient within $[-2^{B-1}, 2^{B-1}-1]$ is selected. Coefficients are rounded and clipped to that range. Bias is independently quantized in Q1.15 with a 19-bit container. Inference then uses the integer dot product and a single arithmetic shift. Candidate widths of 8, 10, 12, 14, and 16 bits are evaluated in the confirmation analysis to show the sensitivity of the readout rather than relying on one development result.

### C. Predictive and Arithmetic Baselines

Four model variants are evaluated.

1) The approximate delayed reservoir is the proposed bit-accurate model.
2) The exact fixed-point reservoir uses the same masks, delay, activation, formats, and training method but exact recurrent products.
3) An 11-tap linear-history model regresses the target on a bias and $u_t, u_{t-1}, \ldots, u_{t-10}$. It tests whether the reservoir adds value beyond inexpensive explicit input memory.
4) A constant predictor uses the fitting-set target mean. Its expected NRMSE is close to one.

The exact reservoir shares hyperparameters selected for the approximate configuration. The area comparison is therefore tightly matched, whereas its accuracy is not claimed to be the best possible exact-reservoir result. Independent exact-model tuning would answer a different question.

Arithmetic behavior is also measured independently of the task. Two hundred thousand signed Q1.15 operand pairs are drawn uniformly using seed 20260721. Approximate and exact shifted products are compared in output least-significant bits (LSBs). Relative error excludes exact products with magnitude below 32 LSBs to avoid unstable ratios near zero. Activation

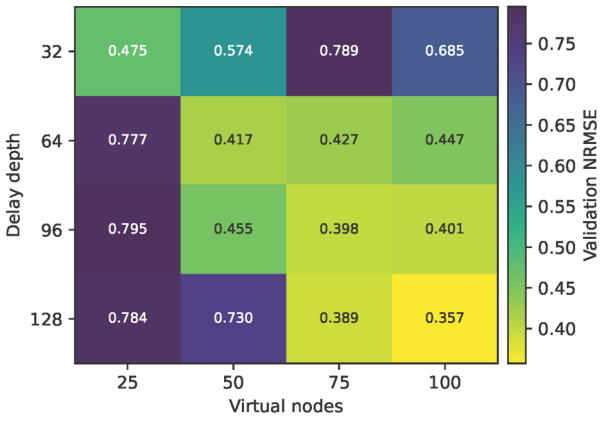



Fig. 3. Development-Sequence Validation NRMSE over the Node and Delay Grid.

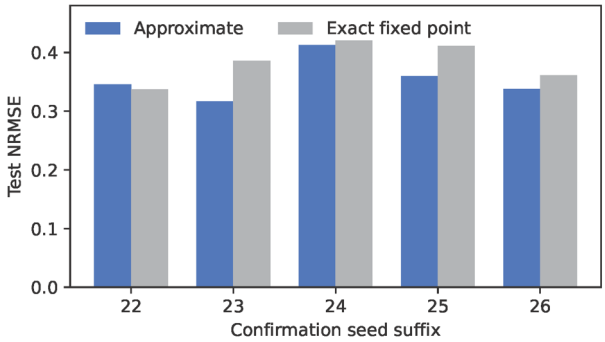



Fig. 4. Approximate and Exact Fixed-Point NRMSE over Five Confirmation Sequences.

error is evaluated over [-4,4] at a spacing of 16 input LSBs against quantized $\tanh$.

### D. Development Design-Space Study

The delay/node grid uses $D \in \{32, 64, 96, 128\}$ and $N \in \{25, 50, 75, 100\}$. Each point is simulated with the locked arithmetic, $\rho$, input scale, leak, mask generator, and 16-bit readout procedure. The readout is fitted on the development training segment and scored on its validation segment. All grid points fit within the 256-word combined memory because $D + N \leq 128$.

This grid is intentionally modest. It verifies that the selected point is not an arbitrary expansion of state and shows the interaction between delay and node count. It is not an exhaustive optimizer over every dynamical parameter. The configuration is frozen after this development analysis; the confirmation results are not used to revisit the grid.

### E. RTL Verification and FPGA Flow

The Python reference uses integer operations matching RTL bit widths, arithmetic shifts, LFSR phase, memory order, and saturation. A self-checking testbench compares eight continuous input transactions to generated deployment vectors. Continuous operation is essential because it verifies that the delay pointer and recurrent state persist correctly between inputs. Component regressions also exhaustively check all 65,536 inputs of the linear-to-log converter retained from the earlier arithmetic work, compare 1,024 generated node transitions, verify wrapper transactions for 1, 16, and 400 updates, and confirm the LFSR period.

Approximate and exact designs are synthesized with Yosys and placed and routed with nextpnr-ice40 [18, 19]. The target is a Lattice iCE40 HX8K in the CT256 package. Both runs use a 100 MHz requested constraint, placement seed 1, and automatic I/O placement. The selected target/package flow has no placeable multiplier blocks, so both configurations map to LUT and carry resources and report zero SB_MAC16 cells. A DSP-enabled synthesis experiment cannot be placed on this target and is not used in the reported comparison. The resulting area comparison therefore applies to LUT-mapped multiplication and does not predict the outcome on a DSP-rich FPGA.

## VII. RESULTS

### A. Development Design Space

Fig. 3 reports validation NRMSE over the node/delay grid. Increasing only one dimension does not produce monotonic improvement. With delay 32, additional nodes beyond 25 worsen validation performance, indicating correlated or poorly aligned features. At delays 96 and 128, larger node sets are more effective. The best evaluated point is $N$=100, $D$=128, with validation NRMSE 0.3573. The neighboring $N$=75, $D$=128 point obtains 0.3893, while $N$=100, $D$=96 obtains 0.4008.

The grid supports two design decisions. First, RAM is not an incidental implementation artifact: the longer delay is associated with lower validation error at the selected node count. Second, the 100 sequential updates are useful because they are coupled through delayed state. Merely increasing the number of scalar updates at a short delay does not guarantee a richer representation.

### B. NARMA-10 Confirmation Results

Table III reports every confirmation run. The approximate reservoir outperforms the linear-history baseline on all five seeds. Its mean NRMSE is 0.3544 with sample standard deviation 0.0359, compared with 0.5160±0.0345 for linear history and 1.0084±0.0148 for the fitting-set mean. The paired approximate-minus-linear difference is -0.1616, with 95% confidence interval [-0.1941, -0.1291].

The exact fixed-point reservoir obtains 0.3833±0.0347. Approximate arithmetic is lower on four confirmation sequences and higher on C1. The paired approximate-minus-exact mean is -0.0288, but its 95% interval [-0.0682, 0.0105] includes zero. The evidence therefore supports comparable accuracy under shared parameters, not a general claim that approximation improves prediction. The lower approximate mean may reflect a change in effective reservoir dynamics or regularization, but five sequences are insufficient to separate those explanations.

Fig. 4 displays the per-sequence comparison, and Fig. 5 shows the first 250 test outputs for C1. Both reservoirs follow

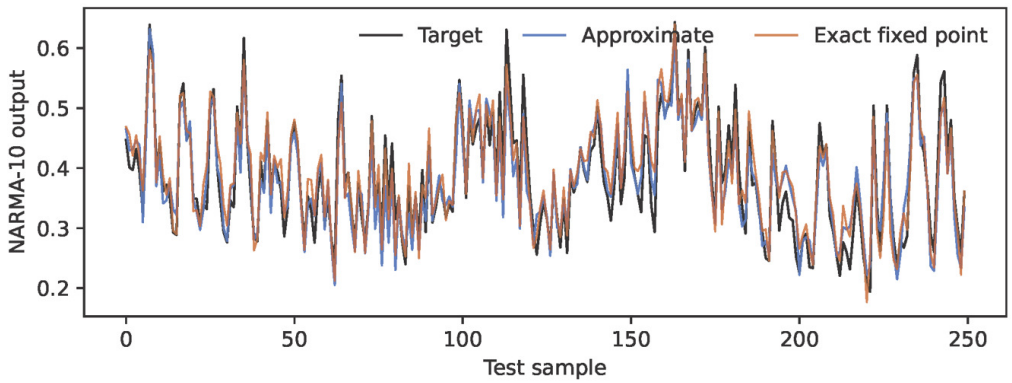



Fig. 5. First 250 Held-Out Test Outputs for Confirmation Sequence C1.

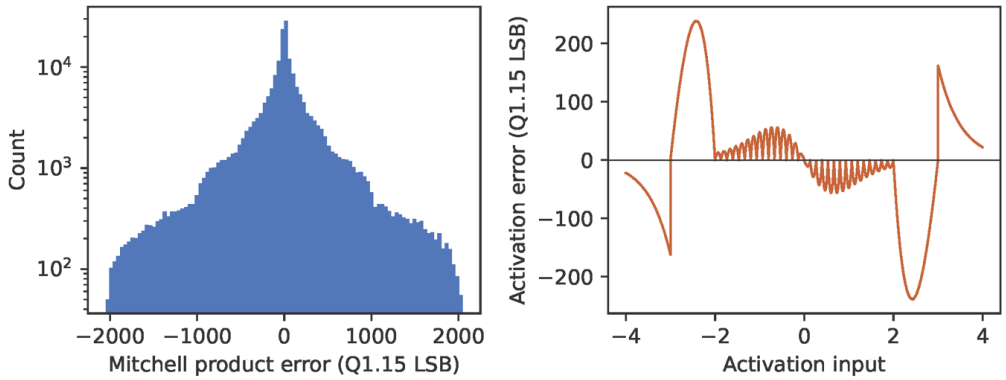



Fig. 6. Mitchell Product and Interpolated Activation Error Characterization.

TABLE III
HELD-OUT NARMA-10 CONFIRMATION NRMSE

| Model | C1 | C2 | C3 | C4 | C5 | Mean | Sample SD |
|---|---|---|---|---|---|---|---|
| Approximate delayed reservoir | 0.3455 | 0.3167 | 0.4124 | 0.3597 | 0.3378 | 0.3544 | 0.0359 |
| Exact fixed-point reservoir | 0.3369 | 0.3861 | 0.4205 | 0.4113 | 0.3615 | 0.3833 | 0.0347 |
| 11-tap linear history | 0.5307 | 0.5098 | 0.5666 | 0.4964 | 0.4765 | 0.5160 | 0.0345 |
| Fitting-set mean | 1.0347 | 1.0015 | 1.0000 | 1.0014 | 1.0043 | 1.0084 | 0.0148 |

the slower target variation and many local peaks, while their errors differ at individual transitions. The trace is illustrative; the numerical conclusions use all 1,000 test samples per sequence.

## C. Arithmetic Error

Table IV characterizes the numerical approximations underlying the preceding task-level result. Across 200,000 random operand pairs, the Mitchell product has mean absolute error 305.36 output LSBs and root-mean-square error 496.07 LSBs. For exact products of at least 32 LSBs, mean absolute relative error is 3.88%, and the 95th percentile is 9.40%. The maximum observed absolute product error is 2,048 LSBs.
These values show that the operator is deliberately approximate rather than a lightly rounded exact multiplier.

The activation approximation is substantially closer to its reference: mean absolute error is 68.50 LSBs and maximum error is 239 LSBs over the evaluated grid. Fig. 6 shows the product-error distribution and the structured activation

TABLE IV
STAND-ALONE ARITHMETIC CHARACTERIZATION

| Metric | Value |
|---|---|
| Random signed product pairs | 200,000 |
| Product mean absolute error | 305.36 LSB |
| Product root-mean-square error | 496.07 LSB |
| Product maximum absolute error | 2,048 LSB |
| Mean absolute relative error[a] | 3.88 |
| 95th percentile relative error[a] | 9.40 |
| Activation mean absolute error | 68.50 LSB |
| Activation maximum absolute error | 239 LSB |

[a]For exact product magnitude $\geq$ 32 LSB.

TABLE V
READOUT-WEIGHT PRECISION

| Weight bits | Mean NRMSE | Sample SD |
|---|---|---|
| 8 | 5.3660 | 0.2902 |
| 10 | 2.1418 | 1.4708 |
| 12 | 0.9113 | 0.2502 |
| 14 | 0.4481 | 0.0604 |
| 16 | 0.3544 | 0.0359 |

TABLE VI
VERIFICATION COVERAGE

| Check | Cases | Result |
|---|---|---|
| Linear-to-log exhaustive | 65,536 | pass |
| Generated node transitions | 1,024 | pass |
| Wrapper | 2 | pass |
| Wrapper | 2 | pass |
| Wrapper | 2 | pass |
| LFSR state period | 65,535 | pass |
| Complete stream | 8 | exact match |

TABLE VII
MATCHED MULTIPLIER-LESS FPGA IMPLEMENTATION RESULTS

| Metric | Approx. | Exact |
|---|---|---|
| LUT4 cells | 2,188 | 2,730 |
| Carry cells | 394 | 362 |
| Flip-flops | 244 | 244 |
| Block RAM | 1 | 1 |
| DSP blocks | 0 | 0 |
| Placed logic cells | 2,407 | 2,959 |
| Placed logic utilization | 31.3% | 38.5% |
| Maximum frequency (MHz) | 23.569 | 24.172 |
| Latency (cycles) | 601 | 601 |
| Latency at $f_{max}$ ($\mu$s) | 25.500 | 24.863 |
| Input rate at $f_{max}$ (k/s) | 39.22 | 40.22 |

interpolation error. These local errors do not map directly to NRMSE because the activation compresses pre-activation magnitude, leakage averages successive states, and the trained readout adapts to the resulting state basis.

### D. Readout Precision Ablation

Table V shows that reservoir-state precision alone is insufficient; the output weights need enough dynamic range and resolution. Eight- and 10-bit weights fail strongly, with mean NRMSE above two. Twelve-bit weights produce mean 0.9113 with large sequence variability, and 14-bit weights improve to 0.4481. The selected 16-bit format reaches 0.3544 and has the smallest across-seed dispersion.

The 16-bit choice costs 1,600 memory bits. Reducing weights to 14 bits would save 200 raw bits, but it would not eliminate the 4-kbit RAM block because the state and weight allocation already occupies a single fixed-capacity block. On this device, the 16-bit format therefore improves accuracy without increasing the number of block memories. A packed multi-model design or an ASIC implementation could present a different precision/storage tradeoff.

### E. Bit-Accurate Verification

Table VI summarizes the executable checks. The complete delayed reservoir matches all eight continuous Python-generated deployment transactions exactly. These transactions cover input latching, mask restart, persistent delayed state, 100 node updates, weight addressing, signed accumulation, bias addition, saturation, and output handshake. The component regressions provide broader arithmetic coverage, although the legacy scalar wrapper is not used as evidence for the new reservoir's task accuracy.

The complete test is intentionally continuous rather than resetting between vectors. A test that reset the delay for every sample could pass despite an incorrect pointer or state-persistence implementation. The generated vectors and combined memory image are produced by the same locked reference script used for confirmation analysis, reducing the risk that simulation exercises different parameters from those reported in the paper.

TABLE VIII
COMPONENT RESOURCE BREAKDOWN (APPROX. VS EXACT)

| Component | LUT4 Cells | | Carry Cells | |
|---|---|---|---|---|
| | Approx. | Exact | Approx. | Exact |
| Readout MAC (Exact) | 972 | 972 | 44 | 44 |
| Activation Unit | 628 | 628 | 72 | 72 |
| Recurrent Multiplier | 383 | 865 | 63 | 29 |
| Controller | 27 | 27 | 6 | 6 |
| Top-level Logic & Routing | 178 | 238 | 209 | 211 |
| Total System | 2,188 | 2,730 | 394 | 362 |

### F. FPGA Resource and Timing Results

Table VII presents the matched iCE40 HX8K results. Approximate multiplication uses 2,188 LUT4 cells, compared with 2,730 for exact multiplication. The reduction is 542 LUT4 cells or 19.85%. The approximate design uses more carry cells, 394 versus 362, because leading-one, fraction, and reconstruction logic shifts the balance from LUT multiplication toward add/shift structures. Both configurations use 244 flip-flops, one block RAM, and no DSP blocks.

To identify which components dominate this implementation cost, we performed an isolated synthesis of the major architectural blocks for both the approximate and exact designs, as detailed in Table VIII . The 16-bit exact readout multiply-accumulate (MAC) unit is the most resource-intensive static component (972 LUT4 cells) because it relies on standard LUT-based multiplication. The piecewise non-linear activation unit consumes 628 LUT4 cells. The primary divergence lies in the recurrent multiplier: the exact LUT-based multiplier requires 865 LUT4 cells, whereas the proposed Mitchell multiplier requires only 383 LUT4 cells. The remaining logic comprises the state accumulator, LFSR mask generator, memory address counters, and top-level multiplexing. This side-by-side breakdown confirms that the exact readout MAC heavily dominates the baseline resource profile, reinforcing the necessity of approximating the recurrent multiplier path to achieve the 19.85% overall area savings. Furthermore, it reveals that the slight increase in total carry cells (from 362 to 394) is entirely localized to the Mitchell multiplier, which leverages carry-chain resources to construct its priority encoders and internal adders, trading a marginal increase in carry cells for a massive reduction in LUTs.

After placement, the approximate design occupies 2,407 of 7,680 logic cells, whereas the exact design occupies 2,959. The reduction is 552 cells or 18.65%. The two implementations have identical algorithmic latency. Approximation does not improve maximum frequency: the approximate build reaches 23.569 MHz and the exact build reaches 24.172 MHz. Corresponding input latencies are 25.500 and 24.863 $\mu$s.

Neither design meets the requested 100 MHz constraint. The approximate critical period is 42.429 ns, and timing analysis places the longest path through activation and state-update

arithmetic rather than a general exact multiplier. Priority-encoder and reconstruction paths also introduce routing pressure. The result is therefore an area improvement, not a speed improvement. A pipeline register between activation and state write could shorten the critical path because the controller already separates logical phases, but that change must be verified carefully to preserve address, LFSR, and state alignment.

## VIII. Discussion

The matched comparison shows that approximate recurrence reduces LUT use on the evaluated multiplier-less FPGA without a statistically resolved accuracy difference from exact recurrence. Both reservoirs remain substantially more accurate than linear input history. Unified memory and immediate accumulation further avoid a separate feature buffer while preserving delayed temporal state and trained readout weights in one block RAM.

However, these hardware benefits are inherently target-specific and represent a critical trade-off when compared to DSP-enabled targets. If this architecture were synthesized on a fabric with available hardened DSP blocks, a conventional exact implementation would map its multipliers directly to those blocks, requiring near-zero LUT utilization for arithmetic. Therefore, the LUT savings of the proposed approximate log-domain design are strictly advantageous only on architectures lacking dedicated multipliers, or in partitions where dedicated DSP slices are fully exhausted. On DSP-rich fabrics, conventional exact multiplication is significantly more resource-efficient. Future work should evaluate board-level energy profiles and additional time-series tasks across these divergent hardware constraints.

## IX. Conclusion

The complete streaming reservoir combines a 128-word delay, 100 masked virtual nodes, approximate recurrence, and an immediate quantized readout in one 4-kbit memory. Across five held-out NARMA-10 sequences it obtains NRMSE 0.35440.0359, versus 0.51600.0345 for linear history and 0.38330.0347 for exact recurrence under the same hyperparameters selected for the approximate design. Python and RTL agree exactly. On the multiplier-less iCE40 HX8K target, approximation reaches 23.569 MHz with 601-cycle latency. The result is a reproducible accuracy–area point for LUT-mapped, ultra-low-resource FPGA inference; board energy, pipelined timing, additional tasks, independently tuned arithmetic, and DSP-equipped targets remain future work.

## References

[1] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 127–149, 2009.

[2] H. Zhang and D. V. Vargas, "A survey on reservoir computing and its interdisciplinary applications beyond traditional machine learning," *IEEE Access*, vol. 11, pp. 81033–81070, 2023.

[3] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. S. Barbosa, "Next generation reservoir computing," *Nat. Commun.*, vol. 12, Art. no. 5564, 2021.

[4] L. Appeltant, M. C. Soriano, G. Van 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," *Nat. Commun.*, vol. 2, Art. no. 468, 2011.

[5] S. Stepney, "Physical reservoir computing: A tutorial," *Nat. Comput.*, vol. 23, pp. 665–685, 2024.

[6] X. Liang, J. Tang, Y. Zhong, B. Gao, H. Qian, and H. Wu, "Physical reservoir computing with emerging electronics," *Nat. Electron.*, vol. 7, pp. 193–206, 2024.

[7] J. N. Mitchell, Jr., "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 4, pp. 512–517, 1962.

[8] Y. Wu, C. Chen, W. Xiao, X. Wang, C. Wen, J. Han, X. Yin, W. Qian, and C. Zhuo, "A survey on approximate multiplier designs for energy efficiency: From algorithms to circuits," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 1, pp. 1–37, 2024.

[9] Z. Ebrahimi, M. Zaid, M. Wijtvliet, and A. Kumar, "RAPID: Approximate pipelined soft multipliers and dividers for high throughput and energy efficiency," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 712–725, 2023.

[10] L. Jin, Z. Liu, and L. Li, "Chain-structure time-delay reservoir computing for synchronizing chaotic signal and an application to secure communication," *EURASIP J. Adv. Signal Process.*, vol. 2022, Art. no. 68, 2022.

[11] S. Ikeda, H. Awano, and T. Sato, "Online training and inference system on edge FPGA using delayed feedback reservoir," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 9, pp. 3323–3335, 2025.

[12] Y. Abe, K. Nishida, K. Ando, and T. Asai, "SPCTRE: Sparsity-constrained fully-digital reservoir computing architecture on FPGA," *Int. J. Parallel Emergent Distrib. Syst.*, vol. 39, no. 2, pp. 197–213, 2024.

[13] Y. Abe, K. Nishida, M. Akai-Kasaya, and T. Asai, "High-order polynomial activation function and regenerative internal weights for FPGA implementation of reservoir computing," *Nonlinear Theory Appl., IEICE*, vol. 15, no. 2, pp. 262–272, 2024.

[14] D. Kleyko, E. P. Frady, M. Kheffache, and E. Osipov, "Integer echo state networks: Efficient reservoir computing for digital hardware," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 4, pp. 1688–1701, 2022.

[15] A. N. Elbedwehy, A. M. El-Mohandes, A. Elnakib, and M. E. Abou-Elsoud, "FPGA-based reservoir computing system for ECG denoising," *Microprocess. Microsyst.*, vol. 91, Art. no. 104549, 2022.

[16] L. Hou, Y. Masuda, and T. Ishihara, "An accuracy reconfigurable vector accelerator based on approximate logarithmic multipliers," in *Proc. 27th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2022, pp. 568–573.

[17] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: Unifying the algorithms and accelerating convergence," *IEEE Trans. Neural Netw.*, vol. 11, no.

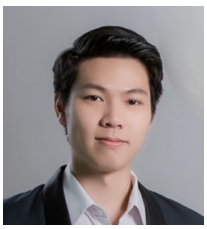
3, pp. 697–709, 2000.
[18] C. Wolf and J. Glaser, "Yosys—A free Verilog synthesis suite," in *Proc. Austrochip Workshop Microelectron.*, 2013, pp. 47–52.
[19] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanović, "Yosys+nextpnr: An open source framework from Verilog to bitstream for commercial FPGAs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2019, pp. 1–4.



**Patinya Ketthong** received the B.Eng. in Computer Engineering from the Thai-Nichi Institute of Technology (TNI), and M.Eng. in ICT for Embedded Systems from Sirindhorn International Institute of Technology (SIIT), Thammasat University, and Ph.D. in Science and Technology from Gunma University, Japan. Since 2018, he has served as a lecturer at TNI. A recipient of the Best Paper Award at the 2016 Management and Innovation Technology International Conference (MITiCON), he has published extensively on chaotic circuits and systems. His research interests include chaotic circuits, chaos-based secure communications, sparse modeling, system identification, and embedded systems.



**Nattagit Jiteurtragool** received the B.Eng. in Computer Engineering from the Thai-Nichi Institute of Technology (TNI), M.Eng. and Ph.D. in Electronics and Photonic Systems Engineering from Kochi University of Technology (KUT), Japan. He is currently an assistant professor at the department of Computer and Information Science, Faculty of Applied Science, King Mongkut's University of Technology North Bangkok, Thailand. His main research interests include the Chaos System, Embedded System, Cryptography, Deep Learning, and Internet-of-Things.



**Chatchai Wannaboon** received the B.Eng. in Computer Engineering from the Thai-Nichi Institute of Technology (TNI), M.Eng. and Ph.D. in Electronics and Photonic Systems Engineering from Kochi University of Technology (KUT), Japan. He is currently a lecturer at the department of Electronics Engineering Technology, College of Industrial Technology, King Mongkut's University of Technology North Bangkok, Thailand. His main research interests include the Applied Machine/Deep Learning, Edge Computing, and System Integration.