Edge-Cell Graph Encoding and Fixed-Point Arithmetic for FPGA Growing Neural Gas

Teuku Zikri Fatahillah*, Raditya Artha Rochmanto*†, Achmad Fahrul Aji*†,
Anhar Risnumawan*‡, Chyan Zheng Siow*§, Naoyuki Kubota*
*Department of Mechanical Systems Engineering, Graduate School of Systems Design
Tokyo Metropolitan University, 6-6 Asahigaoka, Hino, Tokyo 191-0065, Japan
{teuku-fatahillah, rochmanto-artha, achmad-aji, risnumawan-anhar}@ed.tmu.ac.jp, {siow-zheng, kubota}@tmu.ac.jp
†Department of Electrical Engineering, Politeknik Negeri Semarang, Indonesia
‡Mechatronics Engineering Division, Politeknik Elektronika Negeri Surabaya, Indonesia
§Changsha Cultural and Creative Arts Vocational College, China

***Abstract*—We present an FPGA implementation of Growing Neural Gas (GNG) on a Sipeed Tang Nano 9K (Gowin GW1NR-9C) as a bounded-memory hardware substrate for online self-organizing clustering. The design uses an FSM datapath with BRAM-based storage, integer fixed-point arithmetic (Q8/Q16), and a compact *edge-cell graph encoding* that stores the upper-triangular adjacency as 8-bit age+1 values, avoiding dense adjacency matrices and dynamic memory allocation. The complete design is implemented without any soft-core processor or floating-point unit, and is evaluated on Two Moons and Concentric Circles benchmarks. The platform establishes a hardware foundation for deploying continual self-organizing clustering on resource-constrained edge devices.**

***Index Terms*—Growing Neural Gas, self-organizing clustering, FPGA, integer fixed-point arithmetic, unsupervised learning, topology learning, edge computing**

I. Introduction

Intelligent systems deployed on small aerial robots, micro-UAV swarms, and field robots increasingly demand on-board unsupervised adaptation for tasks such as online topology formation, novelty detection, and structure discovery. In these dynamic environments, streaming raw sensory data to a cloud backend is often impractical due to intermittent connectivity and latency, so learning must execute locally with bounded per-sample time and predictable state size. Self-organizing clustering algorithms are well suited to this setting because they can discover data structure incrementally without predefined class labels or offline retraining.

Growing Neural Gas (GNG) is a well-established self-organizing clustering algorithm for topology learning and vector quantization [1]. Building on earlier self-organizing methods [2], [3] and the growing cell structures framework [4], GNG incrementally adapts its graph structure to approximate the input data distribution without retraining cycles. Its sample-by-sample update rule, dynamic node insertion, and edge-aging mechanism make it inherently compatible with incremental learning, where the algorithm continuously refines its representation as new data arrives.

However, deploying GNG at the edge introduces practical challenges: typical embedded targets provide only a few kilobytes of on-chip RAM, lack floating-point hardware, and require deterministic execution timing.

FPGA targets address many of these constraints: they offer custom integer datapaths, synchronous block RAMs (BRAMs) with one-cycle access, and DSP multipliers that execute fixed-width multiplications in a single clock cycle. However, conventional GNG engineering choices remain misaligned with typical small-FPGA constraints. Dense adjacency matrices are particularly mismatched because connectivity storage scales as $O(N^2)$, so even modest caps (e.g., $N_{\max} = 40$) make adjacency dominate the memory footprint: a full 40×40 matrix of 8-bit edge ages requires 1,600 B. Floating-point arithmetic incurs either dedicated FPU blocks or slower iterative units, whereas integer-only datapaths map cleanly to FPGA LUTs and DSP slices.

To address these constraints, we propose an FPGA GNG design that combines compressed *edge-cell* graph storage with 16-bit integer arithmetic and shift-based learning-rate approximations, implemented entirely in synthesizable VHDL as a single-clock FSM on the Sipeed Tang Nano 9K. The design stores node state (coordinates, degree, error) in one BRAM and edge state in a second BRAM, and processes each input sample through a sequential pipeline of read-modify-write operations. Under fixed-capacity allocation ($N_{\max} = 40$), the proposed edge-cell representation reduces the allocated edge-storage from 1,600 B (dense 8-bit $N_{\max}\times N_{\max}$ adjacency) to 780 B (upper-triangular half-adjacency), a 51.25% reduction, while the full GNG state occupies only 1,180 B of BRAM. The main contributions are as follows:

(i) a synthesized FPGA GNG design targeting the Gowin GW1NR-9C FPGA, with integer-only arithmetic and BRAM-mapped node/edge storage;
(ii) an *edge-cell* connectivity encoding storing the upper-triangular adjacency as 8-bit *age+1* values (0 = no edge) to minimize BRAM usage and branch complexity;
(iii) shift-based Q8/Q16 approximations for winner and neighbor learning rates, enabling single-cycle DSP multiplications without a floating-point unit;
(iv) a winner-indexed *fused update pass* that performs error accumulation, error decay, edge aging, neighbor move-

ment, and over-age edge pruning in a single sequential scan, with per-node degree counters for $O(N_{\max})$ isolated-node detection.

These design choices target the constraints of edge robotics and autonomous systems, where deterministic per-sample latency, bounded memory, and integer-only datapaths are essential. The implementation source code developed by the authors is publicly available in a GitHub repository [5].

II. Related Work

Self-organizing clustering algorithms learn topological representations from unlabeled data through competitive learning and local adaptation. SOM [2], [3] and the growing cell structures framework [4] established the foundations, and Fritzke introduced GNG [1] to remove the fixed-topology limitation by allowing the graph to grow and rewire according to the input distribution. Several GNG variants specifically target online and continual learning. Hamker discussed lifelong learning properties [6], Marsland et al. proposed GWR to control growth via novelty criteria [7], Holmström introduced utility-based node management [8], and Prudent et al. developed IGNG for streaming updates [9]. Other work improves scalability through batch, hierarchical, and distributed learning, including Toda et al. [10]–[12], Siow et al. [13], MapReduce-based scaling [14], and multi-scale batch learning for motion extraction [15]. These extensions primarily target large datasets and conventional computing resources.

Self-organizing networks have been implemented on FPGAs using fixed-point arithmetic. Coelho et al. demonstrated an FPGA SOM implementation and highlighted the hardware suitability of competitive learning [16]. Later SOM accelerators focused on throughput via parallel architectures and memory hierarchies [17]–[20], but these designs assume static lattice connectivity and do not address GNG-specific requirements such as sparse dynamic adjacency, edge aging, and bounded-memory growth. FPGA-oriented work has also applied GNG in specific systems and scalable architectures, including a virtual anemometer [21], spike sorting [22], and NoC-style scaling [23]. Drift-adaptive growing approaches further address recurring concept drift via prototype management [24].

For graph-based self-organizing methods, Piyasena et al. [25] implemented a dynamically growing self-organizing neural network (GWR-like) on FPGA for lifelong learning at the edge, demonstrating that structural plasticity can be realized in hardware while meeting real-time constraints. Work on compressed graph representation also provides relevant design principles: Firmli et al. [26] showed that compressed sparse row (CSR++) variants can reduce memory overhead for low-degree graphs, while Zhou et al. [27] demonstrated that adjacency pruning in graph neural networks yields substantial inference speedup with negligible accuracy loss. These results motivate exploring similar integer-arithmetic and memory-aware strategies for graph-based self-organizing methods such as GNG, which share the winner-selection and local adaptation structure of SOM but introduce additional requirements such as edge aging, sparse connectivity, and dynamic growth.

Implementation studies have also optimized GNG for large N by accelerating nearest-neighbor search and reducing bookkeeping. Fišer et al. reported substantial speedups using grid-based partitioning [28]. In contrast, the present work targets small networks ($N \approx 15$–40) under kilobyte-scale BRAM budgets, where memory footprint, integer arithmetic, and deterministic static allocation dominate design choices. In application-driven work, GNG has been adopted for tasks such as landmark learning in mobile robotics [29] and accelerated 3D reconstruction using GPGPU computing [30], demonstrating its practical value in real-world perception and mapping. However, these implementations generally assume workstation-class or GPU-class resources and do not treat memory footprint and arithmetic format as primary constraints. To the best of our knowledge, no prior work combines all of the following in a single synthesized design: (1) a full integer-only GNG datapath on a sub-10K LUT FPGA, (2) half-adjacency age+1 edge-cell encoding for bounded-memory graph storage, (3) a fused update pass that merges error decay, edge aging, and neighbor adaptation into one sequential scan, and (4) deterministic per-sample execution with no soft-core processor.

III. Methodology

A. GNG Algorithm Overview

The Growing Neural Gas algorithm is a self-organizing clustering method that maintains a graph $G = (V, E)$ where V represents nodes with weight vectors $w_i \in \mathbb{R}^d$ and E represents edges between nodes. Unlike batch clustering methods, GNG processes each input sample individually and updates the graph incrementally, making it inherently suitable for online and streaming scenarios. Algorithm 1 presents the core GNG training procedure.

B. FPGA Architecture

1) Overview: The GNG datapath is implemented as a single synthesizable VHDL entity on the Gowin GW1NR-9C. The design uses no soft-core processor: all GNG operations are executed directly by a synchronous FSM running at 27 MHz (crystal, no PLL). Two separate block RAMs store node state and edge state respectively, and input data samples are received from an external host via UART, stored in a third BRAM used as a 100-entry dataset ROM.

2) Block RAM Layout: Node state is packed into 80-bit words stored in a node BRAM with one entry per node:

$$\underbrace{x[15:0]}_{16}\ \underbrace{y[15:0]}_{16}\ \underbrace{\text{act}[0]}_{1}\ \underbrace{\text{deg}[7:0]}_{8}\ \underbrace{\text{err}[31:0]}_{32}$$

where x, y are 16-bit signed integer coordinates in the range $[-1000, 1000]$ (integer-pixel scale after min-max normalization to $[-1, 1]$ and scaling by 1000), `act` is the active flag, `deg` is the 8-bit degree counter, and `err` is the 32-bit unsigned accumulated error. Edge state is stored in a separate BRAM as 8-bit *edge-cells* (age+1 encoding) organized in

Algorithm 1 Growing Neural Gas Training

Initialize two nodes
for each training iteration t **do**
  Sample input ξ from dataset
  Find winner s_1 and second winner s_2:
    $s_1 = \arg\min_i \|\xi - w_i\|^2$
    $s_2 = \arg\min_{i \neq s_1} \|\xi - w_i\|^2$
  Accumulate winner error: err_{s_1} += $d_{s_1}^2 \gg$ ERR_SHIFT
  Decay winner error: err_{s_1} −= $\text{err}_{s_1} \gg$ ERR_DECAY_SHIFT
  Update s_1: $w_{s_1} \leftarrow w_{s_1} + \epsilon_w(\xi - w_{s_1})$
  for each node $n \neq s_1$ (active) **do**
    Decay error: err_n −= $\text{err}_n \gg$ ERR_DECAY_SHIFT
    if edge(s_1, n) exists **then**
      Age edge(s_1, n)
      $w_n \leftarrow w_n + \epsilon_n(\xi - w_n)$
      If over-age and not (s_1, s_2): prune edge, decrement degrees
    end if
  end for
  Create/reset edge (s_1, s_2) with age = 0; increment degrees if new
  Flag isolated nodes (degree = 0) for monitoring (node not deactivated)
  if $t \mod \lambda == 0$ and $|V| < N_{\max}$ **then**
    Let $q = \arg\max_i \text{err}_i$ and $f = \arg\max_{n \in \mathcal{N}(q)} \text{err}_n$
    Insert new node r: $w_r \leftarrow (w_q + w_f)/2$
    Split edge(q, f); add edges (q, r) and (r, f)
    $\text{err}_q \leftarrow \text{err}_q \gg \alpha_{\text{shift}}$; $\text{err}_f \leftarrow \text{err}_f \gg \alpha_{\text{shift}}$; $\text{err}_r \leftarrow \text{err}_q$
  end if
end for

upper-triangular order (Section III-D). Both BRAMs support synchronous read with one-cycle latency, so each read-modify-write sequence requires at least three clock cycles (address, wait, evaluate).

3) FSM Pipeline: The FSM executes the following sequential phases per training iteration:

1) **INIT**: Clear node and edge BRAMs; write two seed nodes at arbitrary initial positions (e.g., $(-500, 500)$ and $(500, -500)$). A soft-reset (`start_i`) can re-enter this phase from any FSM state.
2) **SAMPLE**: Fetch the current data sample (x, y as 16-bit integers) from the dataset BRAM.
3) **WIN_SCAN**: Scan all $N_{\max}$ node slots sequentially; compute $d^2 = (x_s - x_n)^2 + (y_s - y_n)^2$ as a 35-bit unsigned integer and maintain the two nearest nodes (s_1, s_2).
4) **UPDATE**: Read s_1 from node BRAM; accumulate error (err += $d_{s_1}^2 \gg$ `ERR_SHIFT`); apply decay; move winner position using Q8-scaled learning rate; write back.
5) **NB_SCAN**: Scan all $N_{\max}$ node slots; for each active node $n \neq s_1$: (a) apply error decay, (b) read incident edge from edge BRAM, (c) if a neighbor edge exists: age it, move n toward the sample using Q16-scaled rate, prune if over-age.
6) **CONNECT**: Read edge(s_1, s_2); reset to age 0; increment degrees if edge was new.
7) **INSERT** (every λ iterations): Scan all nodes for max-error q; scan neighbors of q for max-error neighbor f; insert midpoint node r; split edge; scale errors.
8) **DBG/SNAP**: Emit per-iteration debug TLV frame (54 bytes, including a 32-bit FPGA cycle-counter timestamp) over UART; every `SNAP_EVERY`= 50 iterations, additionally stream a full node snapshot and active-edge snapshot.

C. Integer Fixed-Point Arithmetic

1) Coordinate Representation: Node coordinates are 16-bit signed integers in $[-1000, 1000]$, obtained by min-max normalizing inputs to $[-1, 1]$ and scaling by 1000.

2) Distance Computation: For active node n with coordinates (x_n, y_n) and sample (ξ_x, ξ_y), the squared Euclidean distance is computed as:

$$\Delta x = \xi_x - x_n, \quad \Delta y = \xi_y - y_n \quad \text{(17-bit signed)}, \tag{1}$$

$$d^2 = \Delta x^2 + \Delta y^2 \quad \text{(35-bit unsigned)}. \tag{2}$$

With 16-bit signed coordinates, the difference fits in 17 bits, each squared difference in 34 bits, and their sum in 35 bits. No overflow is possible, allowing the distance unit to map directly onto the 9×9 DSP multipliers of the GW1NR-9C.

3) Winner Adaptation (Q8 Learning Rate): The winner node s_1 is moved toward the sample using a Q8-scaled learning rate:

$$\Delta x_{s_1} = \left\lfloor \frac{\texttt{EPS\_WIN\_Q8} \cdot (\xi_x - x_{s_1})}{2^8} \right\rfloor, \tag{3}$$

where $\texttt{EPS\_WIN\_Q8} = 77$ corresponds to $\epsilon_w = 77/256 \approx 0.301$. The product $\texttt{EPS\_WIN\_Q8} \times \Delta x$ is a 33-bit intermediate, right-shifted by 8, and the result is added to x_{s_1} with saturation to 16-bit bounds.

4) Neighbor Adaptation (Q16 Learning Rate): Each neighbor node n of s_1 is moved using a Q16-scaled learning rate:

$$\Delta x_n = \left\lfloor \frac{\texttt{EPS\_N\_Q16} \cdot (\xi_x - x_n)}{2^{16}} \right\rfloor, \tag{4}$$

where $\texttt{EPS\_N\_Q16} = 66$ corresponds to $\epsilon_n = 66/65536 \approx 0.001$. The product requires a 35-bit intermediate, right-shifted by 16, giving a small correction that is added to x_n with saturation.

5) Error Accumulation and Decay: Error is accumulated in a 32-bit unsigned register per node:

$$\text{err}_{s_1} \mathrel{+}= d_{s_1}^2 \gg \texttt{ERR\_SHIFT}, \tag{5}$$

where $\texttt{ERR\_SHIFT} = 4$ scales the accumulated error to avoid 32-bit overflow across many iterations. Error decay is applied to every active node during the NB_SCAN pass using a single arithmetic right-shift:

$$\text{err}_n \mathrel{-}= \text{err}_n \gg \texttt{ERR\_DECAY\_SHIFT}, \tag{6}$$

where ERR_DECAY_SHIFT $= 8$ corresponds to $\beta \approx 1 - 2^{-8} = 255/256 \approx 0.9961$ per iteration. This shift-based formulation requires only a single subtraction and eliminates the need for a multiplier in the decay path.

D. Memory Optimization

1) Edge-Cell Graph Encoding: Conventional GNG implementations store edge ages in a dense $N_{\max} \times N_{\max}$ adjacency matrix, which scales as $O(N_{\max}^2)$. For $N_{\max} = 40$ with 8-bit ages, a full matrix requires $40 \times 40 = 1{,}600$ B. Even the upper triangle of a 16-bit matrix ($40 \times 41/2 \times 2 = 1{,}640$ B) is large relative to the available BRAM on a small FPGA.

We store edges in a *compressed half-adjacency* array (upper triangle only), with one byte per potential undirected edge:

$$E_{\text{cells}} = \frac{N_{\max}(N_{\max} - 1)}{2} \text{ bytes} = \frac{40 \times 39}{2} = 780 \text{ bytes}.$$

Each entry uses an **age+1** encoding:

- edge_cell[idx] $= 0$: edge is inactive (no connection),
- edge_cell[idx] $= v > 0$: edge is active with true age $(v - 1)$.

This representation eliminates a separate active flag bit and supports compact over-age deletion using a single threshold comparison (v > A_MAX_STORED, where A_MAX_STORED = A_MAX + 1 = 51).

2) Static BRAM Allocation: The proposed representation is *fixed-capacity*: node and edge BRAMs are allocated once at synthesis time and reused throughout training. This makes memory usage deterministic and eliminates dynamic allocation from the hardware datapath.

To support isolated-node detection without scanning all edges, we maintain a per-node **degree counter** (deg field in the 80-bit node word) that is incremented when a new edge becomes active and decremented when an edge is deleted. A node is isolated if and only if deg[i] == 0.

E. Edge Management Strategy

1) Half-Adjacency Indexing: We store only the upper triangle ($i < j$) of the undirected adjacency. The pair (i, j) is mapped to a linear index:

$$\texttt{idx}(i, j) = \frac{i(2N_{\max} - i - 1)}{2} + (j - i - 1), \quad i < j,$$

with (i, j) swapped if $i > j$. This yields a contiguous edge_cell buffer of size $N_{\max}(N_{\max}-1)/2$ bytes addressable with a 13-bit BRAM address for $N_{\max} = 40$.

2) Fused NB_SCAN Pass: Canonical GNG separates global error decay (an $O(N_{\max})$ sweep) from the winner-incident edge aging and neighbor adaptation (at most $N_{\max} - 1$ edge touches). Because error decay must visit every active node each iteration, we fold the winner-incident edge operations into the *same* pass: while stepping over node slot n to apply its decay, we also probe edge_cell[idx(s_1, n)] and, if the edge is active, handle aging, neighbor movement, and pruning inline. For each active node $n \neq s_1$:

1) Compute decayed error (shift subtraction).
2) Read the edge-cell edge_cell[idx(s_1, n)] from edge BRAM.
3) If the edge is inactive (v == 0): write back only the decayed error.
4) If the edge is active: increment encoded age (v++); apply neighbor adaptation to (x_n, y_n); if v > A_MAX_STORED and the edge is not the (s_1, s_2) connection being refreshed: prune the edge (write 0), decrement degrees.

This fuses *global error decay*, *edge aging*, *neighbor movement*, and *old-edge pruning* into a single sequential pass of at most $N_{\max}$ BRAM read-modify-write sequences, eliminating the separate full-array decay sweep required by a naive implementation. This approach shares the motivation of the lazy error evaluation proposed by Fišer et al. [28], which defers error decay computation entirely and reconstructs the true error on demand using a cycle counter. However, because our design already requires a full $N_{\max}$-slot scan for edge aging and neighbor adaptation, applying the decay inline adds only a single shift-subtract per slot at negligible cost, whereas the lazy scheme would require per-node timestamp storage and variable-length exponentiation that would complicate the fixed-latency FSM datapath.

3) Degree-Assisted Isolated-Node Monitoring: Because the degree counter is maintained in the 80-bit node word, isolated-node detection requires only reading the degree field with no additional edge scan. When a node's degree reaches zero (due to pruning), the iso_flag is set and the node ID is logged in the per-iteration debug frame. Importantly, nodes are *not* deactivated when they become isolated: the node remains active in the BRAM and continues to participate in winner selection. This design choice simplifies the FSM by avoiding cascaded deactivation logic and ensures that isolated nodes can re-acquire edges in future iterations.

F. UART Debug and Visualization Interface

The FPGA streams two types of serial frames at 1 Mbit/s:

- **DBG frame** (every iteration): a 54-byte TLV packet reporting winner IDs, degrees, connection flags, error value, winner position, insertion/pruning events, and a 32-bit FPGA timestamp (cycle count since last soft-reset).
- **Snapshot frames** (every SNAP_EVERY$= 50$ iterations): a fixed-length node snapshot (4 + $N_{\max} \times 7$ bytes) followed by a variable-length active-edge snapshot (4 + $n_e \times 3$ bytes) containing only edge-cells with $v > 0$.

A free-running 32-bit cycle counter, clocked at 27 MHz, is latched into the DBG frame at each sample request, enabling the host to measure per-iteration and total training time directly from the FPGA clock domain. The counter resets on start_i, which also serves as a soft-reset signal: asserting start_i forces the FSM back to the initialization phase from any state, clearing all node, edge, and iteration counters. This allows the host to restart training with a fresh dataset without requiring an FPGA reconfiguration.

A Processing-based host application receives these frames, parses node positions and edge lists, and renders the evolving GNG topology in real time. A companion Python script reads the CSV logs and produces network growth and convergence plots for analysis.

IV. Experimental Setup

A. Datasets

We evaluate the proposed FPGA GNG design on two 2D benchmarks, each with 100 samples:

- **Two Moons**: two interleaving crescent-shaped manifolds, generated with fixed seed (1234), Gaussian noise $\sigma = 0.06$, and shuffled ordering.
- **Concentric Circles**: two concentric rings with Gaussian noise $\sigma = 0.04$ and shuffled ordering.

Both datasets are min-max normalized to $[-1.0, 1.0]$ per axis, scaled by 1000, and rounded to 16-bit signed integers before UART transmission, yielding an integer coordinate range of $[-1000, 1000]$. The exact 100-point CSV files used in this paper are also provided in the accompanying GitHub repository [5] to enable reproducibility.

B. Evaluation Metrics

Following established evaluation practice for topology-learning algorithms [1], [31], we report both representation quality and topology preservation.

1) Quantization Error: Quantization Error (QE) measures the average reconstruction error between samples and their best-matching units (BMUs):

$$\mathrm{QE} = \frac{1}{N} \sum_{i=1}^{N} \left\| x_i - w_{s(i)} \right\|. \tag{7}$$

2) Topological Error: Topological Error (TE) evaluates whether the first and second BMUs of a sample are connected:

$$\mathrm{TE} = \frac{1}{N} \sum_{i=1}^{N} u(x_i), \tag{8}$$

where $u(x_i) = 1$ if the first and second BMUs for x_i are *not* linked by an edge. QE and TE are evaluated on the final node positions received from the FPGA via UART snapshot, using a Python reference script with matching hyperparameters.

3) Hardware Resource Utilization: In addition to QE and TE, we report FPGA synthesis results: LUT, flip-flop, BRAM, DSP, and CLS utilization from the Gowin place-and-route report on the target device (GW1NR-LV9QN88PC6).

C. Implementation Details

Tables I and II summarize the experimental platform and fixed hyperparameter settings.

TABLE I
HARDWARE AND SOFTWARE CONFIGURATION.

Item	Specification
FPGA	Sipeed Tang Nano 9K (Gowin GW1NR-9C)
Part number	GW1NR-LV9QN88PC6/I5
Logic capacity	8,640 LUT4 / 6,693 FF / 26 BSRAM / 10 DSP
Clock	27 MHz (crystal, no PLL)
Toolchain	Gowin EDA V1.9.12.01
Language	VHDL (synthesizable, single entity)
UART baud	1 Mbit/s
Host visualizer	Processing 4 + Python 3.10

TABLE II
GNG HYPERPARAMETERS USED IN ALL EXPERIMENTS.

Parameter	Value	Hardware implementation
$N_{\max}$ (max nodes)	40	node BRAM depth
$A_{\max}$ (max edge age)	50	threshold `A_MAX_STORED`= 51
λ (insertion period)	100	counter `lambda_cnt`
ϵ_w (winner lr)	≈ 0.301	`EPS_WIN_Q8`= 77, shift 8
ϵ_n (neighbor lr)	≈ 0.001	`EPS_N_Q16`= 66, shift 16
α (error split)	0.5	`INS_ALPHA_SHIFT`= 1
β (error decay)	≈ 0.9961	`ERR_DECAY_SHIFT`= 8
ERR_SHIFT (acc. scale)	4	$d^2 \gg 4$ before accumulation

D. Baseline Comparisons

We compare the proposed edge-cell implementation against a dense 8-bit adjacency baseline under identical $N_{\max}$ and hyperparameters, as summarized in Table III. The baseline represents a conventional $N_{\max} \times N_{\max}$ full adjacency matrix layout; the memory footprint comparison (Table V) is analytical, computed from the fixed allocation sizes at identical $N_{\max} = 40$. Because both layouts share the same arithmetic and update logic, no difference in QE or TE is expected; the comparison here focuses on storage cost.

TABLE III
BASELINES USED FOR COMPARISON.

Method	Arith.	Graph storage
Dense adj. (baseline)	16-bit int	$N \times N$ full, 8-bit
Edge-cells (ours)	16-bit int	half-adj., 8-bit age+1

V. Results

A. Hardware Resource Utilization

Table IV reports the place-and-route resource utilization of the full GNG design on the Gowin GW1NR-9C target device, obtained from the Gowin EDA V1.9.12.01.

The design fits within the GW1NR-9C with 62% LUT and 76% BRAM headroom. The 5 DSP blocks handle the distance and adaptation multiplications; the 6 BSRAM instances map to node, edge, and dataset RAMs.

B. Memory Efficiency

Table V compares the allocated BRAM footprint. Replacing the full $N_{\max} \times N_{\max}$ adjacency (1,600 B) with the upper-triangular layout (780 B) yields a 51.25% edge-storage reduction and 41% total reduction (2,000 B $\rightarrow$ 1,180 B).

TABLE IV
FPGA RESOURCE UTILIZATION ON GOWIN GW1NR-LV9QN88PC6 ($N_{\max} = 40$).

Resource	Used	Available	Utilization
LUT / ALU / ROM16	3205	8640	38%
Flip-Flop (FF)	1408	6693	22%
CLS	2225	4320	52%
BSRAM (SDPB)	6	26	24%
DSP	5	10	50%

TABLE V
ALLOCATED BRAM FOOTPRINT COMPARISON (FIXED-CAPACITY, $N_{\max} = 40$).

Implementation	Node storage	Edge storage	Total
Dense adjacency (baseline)	400 B	1600 B	2000 B
Edge-cells (proposed)	400 B	780 B	1180 B
Reduction (edge storage)			**51.25%**
Reduction (total)			**41.0%**

C. Topology Learning Quality

Figure 1 shows the learned GNG topology on Two Moons. The network tracks both crescent-shaped manifolds and forms a connected graph consistent with the topology-preservation objective. Figure 2 shows the training dynamics over 32 epochs: the network grows rapidly in early epochs and both the quantization error and topological error converge as the node count approaches $N_{\max}$.

To evaluate robustness across different data geometries, we also test on the Concentric Circles dataset (100 samples; noise $\sigma = 0.04$). Figure 3 shows the learned topology: the network correctly separates the two rings with distinct node clusters on each circle. Figure 4 shows the training dynamics over 64 epochs, which extend well beyond the point at which $N_{\max}$ is first reached.

TABLE VI
TOPOLOGY LEARNING QUALITY ON FPGA SNAPSHOTS AT SELECTED EPOCHS.

Dataset	Epoch	Nodes	Edges	QE	TE
Two Moons	16	19	22	0.4750	0.00
	32	35	33	0.1832	0.00
	48	40	37	0.1461	0.00
	64	40	34	0.1455	0.01
Concentric Circles	16	19	33	0.6964	0.00
	32	35	36	0.2592	0.00
	48	40	40	0.2000	0.04
	64	40	39	0.1960	0.07

Table VI reports QE and TE computed offline from the UART node snapshots at epochs 16, 32, 48, and 64. During the growth phase (epochs 16–32), QE drops sharply while TE stays at 0.00 on both datasets. Once $N_{\max} = 40$ is reached (epoch 48 onwards), QE is near its minimum but the two datasets diverge on TE: Two Moons holds at 0.00–0.01, whereas Concentric Circles drifts from 0.04 to 0.07. The drift arises when samples near the inter-ring boundary select their first and second BMUs from different rings, which are not connected by an edge due to the absence of links across the inter-ring gap.

Because our implementation maintains a fixed node budget ($N_{\max} = 40$), learning continues even after the network has saturated: winner-triggered adaptation still moves node positions, and edge aging still inserts and removes links each iteration. Empirically, QE and TE behave differently in this regime: QE continues to decrease overall because the winner-move step ϵ_w still pulls each BMU closer to its assigned samples, reducing mean quantization distance. TE, in contrast, tends to *increase* slowly once $N_{\max}$ has been reached, since node positions drift under continuous adaptation and edge aging removes links faster than the saturated network can re-establish correct neighborhood relationships, as no additional nodes can be inserted to refine local topology. This behavior is visible in the right panels of Figures 2 and 4 and motivates early-stopping or utility-based node management as a direction for future work.

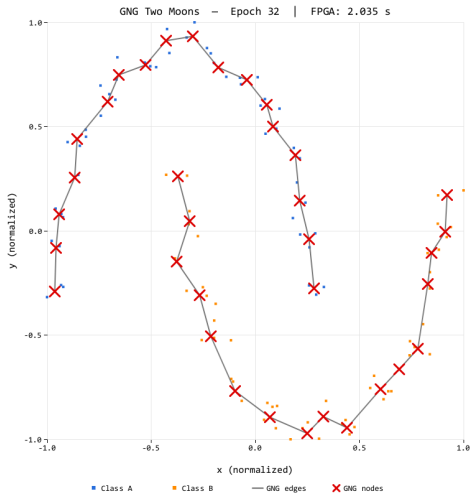

Fig. 1. Learned GNG topology on Two Moons (100 samples; $N_{\max} = 40$). Node positions received via UART and plotted in normalized $[-1, 1]$ coordinates.

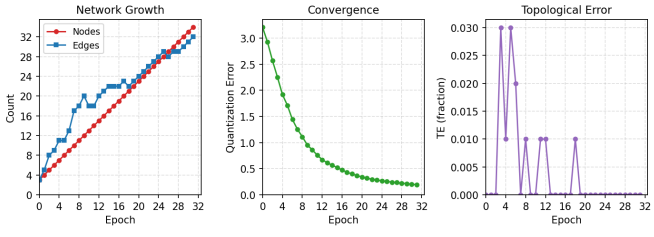

Fig. 2. Two Moons training dynamics over 32 epochs (100 samples/epoch). Left: node and edge count growth. Middle: quantization error convergence. Right: topological error vs. epoch.

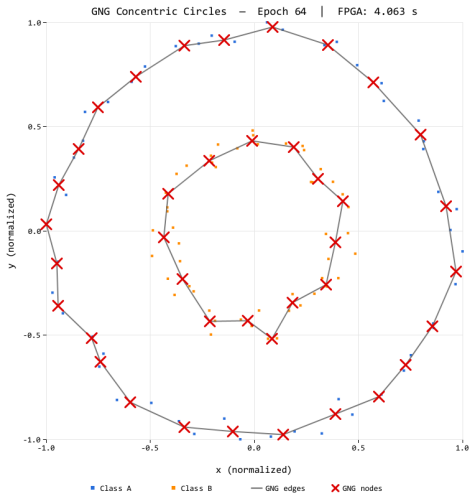

Fig. 3. Learned GNG topology on Concentric Circles (100 samples; $N_{\max} = 40$).

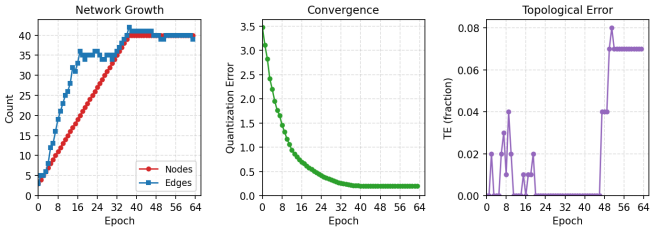

Fig. 4. Concentric Circles training dynamics over 64 epochs (100 samples/epoch). Left: network growth. Middle: quantization error convergence. Right: topological error vs. epoch.

D. Computational Performance

Table VII reports per-sample latency and throughput, measured directly from the 32-bit cycle counter embedded in each DBG frame (Section III-B) and converted using the 27 MHz clock. Per-sample latency is the cycle difference between consecutive frames; total training time is the span from the first to the last frame. Both datasets exhibit consistent latency of $\sim$625 μs ($\sim$16,900 cycles), yielding $\sim$1,600 samples/s. The latency is dominated by the WIN_SCAN and NB_SCAN passes, each requiring $N_{\max}$ sequential BRAM read-modify-write sequences, and is further gated by the per-sample UART debug transmission; the FSM computation alone completes in fewer cycles. This deterministic, bounded latency confirms that the design meets real-time constraints for sample-by-sample processing at moderate data rates.

TABLE VII
FPGA RUNTIME PERFORMANCE AT 27 MHz ($N_{\max} = 40$).

Dataset	Cycles/sample	μs/sample	Samples/s	Total (s)
Two Moons	16,852	624.1	1,602	5.40
Concentric Circles	16,931	627.1	1,595	5.72

VI. Discussion

A. Hardware Trade-offs

1) Sequential FSM vs. Parallel Datapath: The sequential FSM minimizes LUT usage at the cost of $O(N_{\max})$ per-sample latency. A parallel winner-search tree could reduce this to $O(\log N_{\max})$ at the expense of additional LUTs.

2) Integer Approximation of Learning Rates: The Q8 and Q16 shift-scaled learning rates ($\epsilon_w \approx 0.301$, $\epsilon_n \approx 0.001$) are hardware-friendly but not exact. A winner rate of 0.3 rather than the canonical 0.05 causes faster convergence but may slightly increase quantization error. A finer Q8 approximation (e.g., `EPS_WIN_Q8`$=13$ for ≈ 0.05) can be substituted without circuit changes. Despite these approximations, the final QE and TE reported in Table VI remain low on both benchmarks, indicating that the shift-based rates are acceptable for the tested datasets.

3) Isolated-Node Policy: Canonical GNG deactivates a node when it loses all edges. Our FSM instead retains isolated nodes (`deg`$=0$) in BRAM and continues to include them in winner selection. This avoids cascaded deactivation logic and allows the node to re-acquire edges in subsequent iterations. The trade-off is that an isolated node occupies one of the $N_{\max}$ slots without contributing to topology coverage until it regains connectivity.

4) Error Accumulation Scaling: The `ERR_SHIFT`$=4$ factor prevents 32-bit overflow; for larger coordinate ranges or longer training, this parameter should be increased.

B. Relation to Prior FPGA Self-Organizing Work

Prior FPGA SOM designs [16], [17] assume static lattice connectivity. GNG requires per-iteration edge read-modify-write on a variable-structure graph, motivating the BRAM-indexed edge-cell representation. Our results show that full self-organizing clustering with dynamic topology is feasible on a sub-10K-LUT device.

C. Relation to GNG Variants and Continual Learning

This work follows the core update structure of canonical GNG [1] while introducing hardware-oriented approximations to keep the design synthesizable and comparable. The algorithm inherently supports incremental adaptation through error-driven growth and age-based pruning, properties relevant to continual learning. Extensions such as GWR [7], utility-based management [8], and drift-adaptive approaches [24] could benefit from similar edge-cell and integer-arithmetic design.

D. Limitations and Future Works

The current evaluation uses two static 2D benchmarks; validation on non-stationary streaming data remains future work. The present results therefore demonstrate that the hardware substrate supports correct topology learning, but do not yet confirm adaptation under distribution shift.

Future work targets four directions: (i) *continual learning on FPGA*—extending the design to accept streaming sensor data and adapt continuously without restarting; (ii) *parallel winner*

search—replacing the sequential scan with a pipelined tree to reduce latency to $O(\log N_{\max})$; (iii) *configurable precision*—parameterizing Q-format constants via generics; and (iv) *multi-stream input*—accepting real-time sensor data for on-board unsupervised mapping.

VII. Conclusion

This paper presented an FPGA implementation of GNG on the Sipeed Tang Nano 9K (Gowin GW1NR-9C) as a bounded-memory substrate for online self-organizing clustering. The edge-cell graph encoding halves edge-storage relative to a dense adjacency baseline, and a fused NB_SCAN pass combines error decay, edge aging, and neighbor adaptation into a single sweep. On Two Moons and Concentric Circles, the integer-only datapath preserves good topology learning quality with deterministic per-sample latency, showing that Q8/Q16 fixed-point arithmetic and shift-based learning rates are sufficient on the evaluated datasets. These results demonstrate that self-organizing clustering can be deployed on sub-10K-LUT FPGAs when memory layout and arithmetic format are treated as first-class design concerns. Because GNG processes samples incrementally with bounded state, the proposed platform provides a hardware foundation for continual learning on resource-constrained edge devices.

Acknowledgment

This work was supported by the Japan Science and Technology Agency (JST), Moonshot R&D Program, under grant number JPMJMS2034.

The authors used Anthropic Claude Opus 4.6 to refine wording and clarity. All technical content and results were verified by the authors, who take full responsibility for the paper.

References

[1] B. Fritzke, "A growing neural gas network learns topologies," in *Advances in Neural Information Processing Systems*, vol. 7. Cambridge, MA: MIT Press, 1994, pp. 625–632.

[2] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[3] T. Kohonen, *Self-Organizing Maps*, 1st ed. Berlin, Germany: Springer, 1995.

[4] B. Fritzke, "Growing cell structures—a self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 9, pp. 1441–1460, 1994.

[5] T. Z. Fatahillah, "fpga_gng: Source code for embedded and FPGA growing neural gas," GitHub repository, 2026, accessed: Apr. 14, 2026. [Online]. Available: https://github.com/tzf230201/fpga_gng

[6] F. H. Hamker, "Life-long learning cell structures—continuously learning without catastrophic interference," *Neural Networks*, vol. 14, no. 4–5, pp. 551–573, 2001.

[7] S. Marsland, J. Shapiro, and U. Nehmzow, "A self-organising network that grows when required," *Neural Networks*, vol. 15, no. 8–9, pp. 1041–1058, 2002.

[8] J. Holmström, "Growing neural gas with utility," *Neural Networks*, vol. 15, no. 8–9, pp. 1029–1037, 2002.

[9] Y. Prudent and A. Ennaji, "An incremental growing neural gas learns topologies," in *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2005, pp. 1211–1216.

[10] Y. Toda and N. Kubota, "Extraction of multiple topological structures using growing neural gas," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Taipei, Taiwan: IEEE, 2006, pp. 3051–3056.

[11] Y. Toda and N. Kubota, "Batch-learning growing neural gas for fast topological mapping," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Montreal, QC, Canada: IEEE, 2007, pp. 2565–2570.

[12] Y. Toda and N. Kubota, "Multi-layer growing neural gas for hierarchical topological extraction," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Singapore: IEEE, 2008, pp. 2194–2199.

[13] C. Z. Siow, A. A. Saputra, T. Obo, and N. Kubota, "Distributed batch learning of growing neural gas for quick and efficient clustering," *Mathematics*, vol. 12, no. 12, p. 1909, 2024.

[14] E. Ventocilla *et al.*, "Scaling growing neural gas on large data sets using MapReduce and S2 geometry," *Neurocomputing*, vol. 456, pp. 14–31, 2021.

[15] E. Ardilla *et al.*, "Multi-scale batch-learning growing neural gas efficiently for dynamic data distributions," *International Journal of Automation Technology*, vol. 17, no. 3, pp. 206–216, 2023.

[16] P. Coelho, C. Silva, and B. Ribeiro, "FPGA implementation of self-organizing maps using fixed-point arithmetic," *Journal of Signal Processing Systems*, vol. 92, no. 3, pp. 311–326, 2020.

[17] J. Dias *et al.*, "Full-parallel implementation of self-organizing maps on fpga," *Neural Networks*, vol. 140, pp. 392–401, 2021.

[18] L. de Sousa *et al.*, "SOMProcessor: A high throughput FPGA architecture for self-organizing maps applied to video processing," *Neural Networks*, vol. 125, pp. 349–362, 2020.

[19] S. Yamagiwa, Y. Kawahara, A. Kanazawa, and M. Yasunaga, "A highly scalable self-organizing map accelerator on FPGA and its performance evaluation," *Artificial Life and Robotics*, vol. 29, pp. 94–100, 2024.

[20] H. Hikawa, "Place-and-route analysis of fpga implementation of nested hardware self-organizing map architecture," *Electronics*, vol. 12, no. 21, p. 4523, 2023.

[21] G. La Tona, M. Luna, M. C. Di Piazza, M. Pucci, and A. Accetta, "Development of a high-performance, FPGA-based virtual anemometer for model-based MPPT of wind generators," *Electronics*, vol. 9, no. 1, p. 83, 2020.

[22] M. Shoaran, C. Pollo, A. Schmid, and Y. Leblebici, "Hardware-efficient growing neural gas for spike sorting in implantable interfaces," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 6, pp. 847–860, 2015.

[23] F. Derue, A. Dupuis, O. Sentieys, S. Lescouet, and G. Sassatelli, "SWAP-GNG: A self-adaptive architecture for continual learning," in *2025 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. Bordeaux, France: IEEE, 2025, tODO: add pages and DOI if available.

[24] M. Arostegi, M. N. Bilbao, J. L. Lobo, and J. Del Ser, "AiGAS-dEVL-RC: An adaptive growing neural gas model for recurrently drifting unsupervised data streams," in *2025 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2025, also available as arXiv:2504.05761.

[25] D. Piyasena, M. Thathsara, S. K. Lam, and M. Wu, "Dynamically growing neural network architecture for lifelong deep learning on the edge," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 262–268.

[26] A. Firmli *et al.*, "CSR++: Improving the performance of the compressed sparse row format for sparse matrix vector multiplication," in *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 184. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, pp. 17:1–17:16.

[27] Y. Zhou *et al.*, "Accelerating graph neural network inference via channel pruning," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2664–2677, 2021.

[28] D. Fišer, J. Faigl, and M. Kulich, "Growing neural gas efficiently," *Neurocomputing*, vol. 104, pp. 72–82, 2013.

[29] D. Hernández, V. Caselles, and A. L. Elías, "Automatic landmark extraction for mobile robot navigation using growing neural gas," in *Machine Vision Applications*, 2005, tODO: verify full proceedings title and pages.

[30] S. Orts, J. Garcia-Rodriguez, D. Viejo, M. Cazorla, and V. Morell, "Gpgpu implementation of growing neural gas: Application to 3d scene reconstruction," *Journal of Parallel and Distributed Computing*, vol. 72, no. 10, pp. 1361–1372, 2012.

[31] T. Martinetz and K. Schulten, "Topology representing networks," *Neural Networks*, vol. 7, no. 3, pp. 507–522, 1994.