

# Scenario-Aware Control of Segmented Ladder Bus: Design and FPGA Implementation

Phu Khanh Huynh, Francky Catthoor, and Anup Das

**Abstract**—Large-scale neuromorphic architectures consist of computing tiles that communicate spikes using a shared interconnect. The communication patterns in these systems are inherently sparse, asynchronous, and localized, as neural activity is characterized by temporal sparsity with occasional bursts of high traffic. These characteristics require optimized interconnects to handle high-activity bursts while consuming minimal power during idle periods. Among the proposed interconnect solutions, the dynamic segmented bus has gained attention due to its structural simplicity, scalability, and energy efficiency. Since the benefits of a dynamic segmented bus stem from its simplicity, it is essential to develop a streamlined control plane that can scale efficiently with the network. In this paper, we present a design methodology for a scenario-aware control plane tailored to a segmented ladder bus, with the aim of minimizing control overhead and optimizing energy and area utilization. We evaluated our approach using a combination of FPGA implementation and software simulation to assess scalability. The results demonstrated that our design process effectively reduces the control plane’s area footprint compared to the data plane while maintaining scalability with network size.

**Index Terms**—Segmented Bus, Ladder Bus, Neuromorphic Computing, Spiking Neural Networks, Optimization

## I. INTRODUCTION

Neuromorphic computing systems [1] aim to replicate the computational principles of the brain by implementing spiking neural networks (SNNs) in hardware. These systems process information through discrete spike events rather than continuous signal transmission, resulting in a fundamentally different communication pattern compared to traditional computing architectures. Specifically, neuromorphic communication is sparse, asynchronous, and localized [2], where neurons generate spikes only when their membrane potential crosses a threshold. This event-driven behavior leads to temporal sparsity, meaning that most of the network remains idle for extended periods, with occasional bursts of activity when multiple neurons spike simultaneously. Consequently, the interconnect infrastructure in neuromorphic architectures must be optimized to accommodate this unique communication pattern, ensuring efficient data transfer during bursts while consuming minimal power during idle phases.

To facilitate communication in large-scale neuromorphic systems, many-core architectures are often employed, where processing elements (PEs) are interconnected using shared communication fabrics such as buses and networks-on-chip

P. K. Huynh and A. Das are with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA, 19104. E-mail: {ph434, anup.das}@drexel.edu. F. Catthoor is with National Technical University of Athens, Greece. Email: catthoor@microlab.ntua.gr.

Manuscript received Month DD, Year; revised Month DD, Year.

(NoCs) [3], [4]. While NoCs provide structured and scalable communication, they introduce significant overhead due to packet buffering, routing table management, and static link activity [5], [6]. These factors make NoCs less ideal for neuromorphic workloads, where traffic is sporadic. Instead, segmented bus architectures [7] have been proposed as a more efficient alternative due to their ability to dynamically allocate communication pathways based on demand. By dividing the bus into segments and activating only the necessary paths for spike transmission, segmented buses can significantly reduce energy consumption while maintaining low latency.

Among various segmented bus implementations [8], the segmented ladder bus [9] has been shown to offer the best energy efficiency and area utilization for runtime-configurable spiking traffic. This dynamic segmentation balances adaptability, energy efficiency, and scalability, making it well-suited for neuromorphic applications. However, its effectiveness depends on an efficient control mechanism. Because the routing overhead is pushed fully to the control plane, a poorly designed controller can cause unnecessary state transitions, increasing area and power consumption and undermining the benefits of using a dynamic segmented bus. To address this challenge, we are the first to propose and design a system scenario-aware control plane specifically for the segmented ladder bus architecture (to the best of our knowledge). This control plane is tailored to minimize control overhead and improve energy efficiency [10]. By analyzing the traffic pattern of the application, our control plane can be adapted to minimize the frequently occurring communication scenarios. Essentially, by minimizing the number of active control scenarios, our design reduces the control related memory storage size and hence also reduces area footprint of the control plane to a negligible amount of the data plane. This makes it fully suitable for large-scale neuromorphic deployments.

To validate our approach, we implement the essential parts of the proposed control plane on FPGA and perform a combination of hardware-based and software-based simulations. We evaluate the impact of our scenario-aware optimization on area utilization and scalability. Experimental results demonstrate that our control plane design achieves significant reductions in resource overhead while maintaining reliable spike transmission.

The rest of this paper is structured as follows: Section II provides an overview of segmented ladder bus architecture and the existing deployment process for SNNs on such interconnect hardware. Section III introduces our scenario-aware control methodology and optimization techniques. Section IV presents experimental results and analysis. Finally, the paper is concluded in Section V.

## II. BACKGROUND

This section presents the requisite background to comprehend the segmented ladder bus [9]. Furthermore, we review related works on the mapping of SNNs to neuromorphic hardware architectures, emphasizing the associated challenges and recent innovations in this domain.

### A. Segmented Ladder Bus

The architecture of a segmented ladder bus is designed with the following characteristics:

- The tiles are logically divided into two parallel rows, each comprising an equal number of tiles.
- A fixed number of parallel segmented bus lanes are placed between these tile rows.
- The tiles and parallel bus lanes are connected using criss-cross three-way segmented switches.

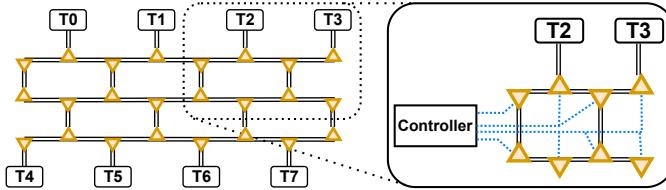


Fig. 1: Segmented ladder bus example with a switch controller.

Figure 1 shows a segmented ladder bus with 8 tiles and 3 bus lanes, which can be scaled to meet application needs. This design offers several advantages, including high flexibility that enables communication between any pair of tiles, improved routing and fault-tolerance through multiple communication paths, run-time configurability allowing dynamic switch reconfiguration under congestion, and enhanced energy and area efficiency due to its bufferless architecture.

A segmented ladder bus uses a compile-time software framework alongside coarse-grain runtime hardware controllers for dynamic switch configuration. These distributed local controllers manage predefined switch scenarios and transmit control signals within their regions. In large-scale networks, many such controllers coordinate switch operation across the system. A key challenge of this dynamic approach is that it shifts all routing overhead to the control plane, making its design crucial for overall efficiency.

### B. Mapping and Scheduling SNNs on Segmented Ladder Bus

In our previous work [11], we have described the SNN application partitioning process in more details. Once an SNN application has been partitioned into clusters, in order to deploy it to a segmented ladder bus, we need to: map the clusters to actual hardware tiles; schedule spike traffic; and route the scheduled traffic. Each of these steps can be optimized to improve the hardware utilization of the network, guarantee performance of the application, and reduce energy consumption [12]. Figure 2 shows this process which is essential for running applications on segmented ladder bus.

This three step process can be summarized as follows:

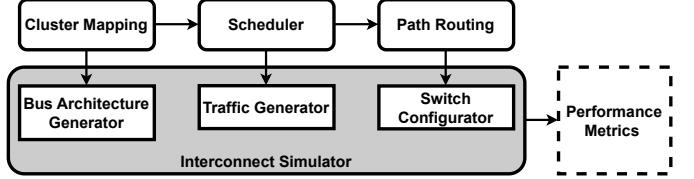


Fig. 2: Mapping and scheduling design flow.

- **Mapping:** The cluster mapping problem in neuromorphic systems involves assigning neuron clusters to hardware tiles. Given the constraints of a segmented ladder bus, the mapping must minimize dynamic energy consumption and reduce intersecting communication paths to prevent delays or packet loss.
- **Scheduling:** Required when simultaneous connections exceed interconnect capacity, scheduling spreads traffic over time to prevent congestion while minimizing additional delays.
- **Routing:** Optimal routing minimizes energy and delay by guiding data through efficient paths. In bufferless segmented switches, it also prevents path crossings that cause collisions and spike loss. Additionally, routing balances traffic for better hardware utilization and provides fault tolerance through alternate paths.

## III. PROPOSED CONTROL PLANE SOLUTION

The control plane of a segmented ladder bus must coordinate dynamic switch configurations in real time based on predefined communication patterns. Without optimization, the number of switching scenarios can grow rapidly with network size, increasing memory and hardware overhead. This motivates a scenario-aware control methodology to manage control information efficiently. The next section presents our proposed control plane design and algorithms, which minimize switching scenarios while ensuring correct communication.

### A. Local Controller Hardware Design

The hardware implementation of a distributed local controller for segmented bus is designed to be simple and efficient. It operates using a local memory bank that stores predefined system scenarios tailored to the specific configuration of the bus segments it manages. During run-time, the controller dynamically selects the appropriate scenario based on the current communication demands. This scenario-based mechanism, identified at design-time and deployed at runtime, supports dynamic operation with a limited number of scenarios. This mechanism works similarly to a distributed loop nest counter [13]. The central software control framework loads the entire loop nest of control scenarios into memories of local controllers. The scenarios are embedded as instructions for both regular (e.g., predictable switch sequences in static traffic patterns) and irregular (e.g., conditional switch reconfiguration triggered by runtime events or spike activity) loop structures. After loading, each controller autonomously steps through local control scenarios using a loop counter. Once a scenario is chosen, the controller generates and transmits the corresponding control signals to the relevant switches.

By issuing the control signals from the controllers with the correct timing, the switches can be used to create paths that support multiple simultaneous connections inside the network, matching application requirements.

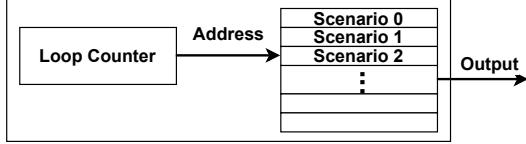


Fig. 3: Controller hardware design.

### B. Control Plane Scenario Design

Using the cluster mapping data, we extract the required communication paths and group them into non-intersecting scenarios to minimize the total number of switching cases. We first apply a greedy algorithm (Algorithm 1) that iteratively assigns each path to the first group where it does not intersect with others, forming a new group if needed. This continues until all paths are assigned, with each group forming a distinct scenario for the control plane. While this ensures conflict-free grouping, it is non-optimal and still produces a relatively high number of scenarios, as shown in Section IV.

Alternatively, we design a more exploratory algorithm that

#### Algorithm 1: Greedy Grouping

---

**Input:** Cluster mapping data with communication paths  
**Output:** Set of non-conflicting switching scenarios

- 1 Initialize scenario set  $S \leftarrow \emptyset$ ;
- 2 **foreach** path  $p$  in the path list **do**
- 3     placed  $\leftarrow$  false;
- 4     **foreach** scenario  $s \in S$  **do**
- 5         if no path in  $s$  intersects with  $p$  **then**
- 6             Add  $p$  to  $s$ ;
- 7             placed  $\leftarrow$  true;
- 8             **break**;
- 9     **end**
- 10    if not placed **then**
- 11      Create a new scenario  $s_{\text{new}} \leftarrow \{p\}$ ;
- 12      Add  $s_{\text{new}}$  to  $S$ ;
- 13    **end**
- 14 **end**
- 15 **return**  $S$ ;

---

#### Algorithm 2: Max Clique Grouping

---

**Input:** Cluster mapping data with communication paths  
**Output:** Set of non-conflicting switching scenarios

- 1 Construct graph  $G(V, E)$  where vertices  $V$  represent communication paths;
- 2 **foreach**  $(p_i, p_j) \in V \times V$  **do**
- 3     if  $p_i$  and  $p_j$  intersect **then**
- 4         Add edge  $(p_i, p_j)$  to  $E$ ;
- 5     **end**
- 6 **end**
- 7 Initialize scenario set  $S = \emptyset$ ;
- 8 **while**  $V \neq \emptyset$  **do**
- 9     Find the maximal clique  $C \subseteq V$ ;
- 10    **foreach**  $v \in C$  **do**
- 11      Assign  $v$  to a distinct scenario in  $S$ ;
- 12    **end**
- 13    Remove  $C$  from  $G$ ;
- 14    if any remaining vertices cannot be assigned to existing scenarios **then**
- 15      Create a new scenario;
- 16    **end**
- 17 **end**
- 18 **return**  $S$ ;

---

reduces greediness and takes into account a broader search space. It works as follows: Communication paths can be represented as vertices in a graph, where an edge is established between any two vertices if their corresponding paths intersect.

The lower bound for the number of required scenarios is determined by the maximal clique of this conflict graph, as all paths within this clique mutually intersect and must therefore be assigned to separate scenarios. To achieve this, a computationally efficient algorithm is employed to identify the maximal clique. The vertices within this clique are then distributed into distinct scenarios. Subsequently, the identified clique is removed from the graph, and the process iterates: identifying the next maximal clique, partitioning its vertices into scenarios while ensuring no path intersections within a single scenario, and repeating the cycle. If, at any stage, the remaining vertices cannot be accommodated within the existing scenarios, a new scenario is created. This iterative process continues until all vertices have been removed.

**Time Complexity Analysis of Algorithm 2:** Let  $n$  be the number of communication paths and  $m$  the number of edges in the intersection graph  $G(V, E)$ . Constructing  $G$  takes  $O(n^2)$  time by checking all path pairs. The main loop runs up to  $O(n)$  times, and each iteration performs maximal clique extraction—an NP-hard task with worst-case time  $O(3^{n/3})$  using exact algorithms like Bron–Kerbosch [14], with graph updates  $O(n+m)$  being negligible in comparison. The overall complexity is  $O(n^2 + n \cdot 3^{n/3})$ .

## IV. RESULTS AND DISCUSSIONS

Table I reports the machine learning applications utilized in the assessment of designing scenario-aware control plane for segmented ladder bus. For each model, we provide detailed information on the total number of clusters, the average connection degree, network density (the connections between clusters are directional), and the largest connection degree. Six realistic and four synthetic applications were used.

Application	# Clusters	Avg. Degree	Network Density	Largest Degree
mnist	11	1.64	0.135	6
LeNet [15]	14	2.93	0.225	11
fashion-mnist [16]	24	5.33	0.230	8
cifar10	26	5.44	0.210	8
emnist [17]	30	5.37	0.185	8
synth_40 (160)	40	4.00	0.105	7
synth_40 (292)	40	7.30	0.185	14
synth_60 (348)	60	5.80	0.100	12
synth_60 (772)	60	12.87	0.220	21
ResNet [18]	96	11.13	0.115	78

TABLE I: Applications used to evaluate control plane scaling.

**FPGA implementation:** To estimate control plane resource usage, we implemented the small and medium applications on FPGA with their respective number of clusters. The segmented ladder bus was configured with a number of lanes equal to the square root of the tile count, each 32 bits wide. Scenario counts were derived using Algorithm 2. The results are presented in Table II. From this table, we can see clearly that the control plane utilizes much fewer resources than the data plane. In all applications, the control plane utilize less than 10% of the total resources, with an average utilization of 6.5%. These practical results show that, in small to medium-scale applications, the control overhead remains minimal compared to the data plane.

**Simulation scalability analysis:** For larger networks that exceed available FPGA resources, we perform an analytical evaluation of the control plane. Specifically, we examine how the number of scenarios—a key scaling parameter—grows

App	# Lanes	# Scen.	D-Plane CLBs	C-Plane CLBs	C-Plane Util.
mnist	3	8	3277	73	2.18%
LeNet	4	13	3503	204	5.50%
fashion mnist	5	24	5722	543	8.67%
cifar10	5	23	6416	548	7.87%
emnist	5	26	7247	645	8.17%

TABLE II: Data and control plane utilizations.

relative to the data plane. Figure 4 compares the total number of connections with the number of scenarios produced by Algorithm 2. The results show that scenario count grows much more slowly than both the number of connections and clusters. In contrast, the data plane scales proportionally with the number of clusters and the number of bus lanes. This indicates that as the network size increases, the control plane area remains within a manageable range and grows more slowly than the data plane, making it negligible in area and energy overhead. As the network scales, each scenario encodes increasingly sparse information with many zero values, creating opportunities for compression that further reduce control memory size. Moreover, scenarios are distributed across local controllers, each handling fewer switches and storing smaller scenario sets, which enhances control plane scalability.

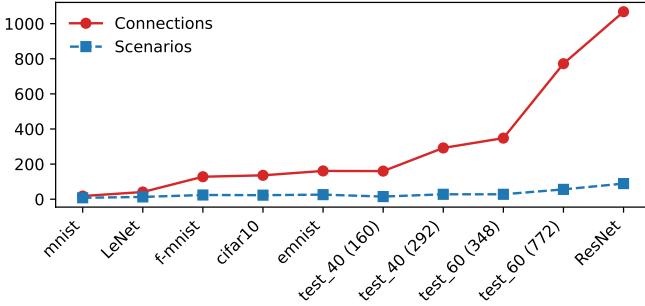


Fig. 4: Number of connections and scenarios scaling.

We implemented both the greedy and maximum clique scenario grouping algorithms, with results shown in Figure 5. The maximum clique approach consistently yields fewer scenarios than the greedy method. The largest node degree provides a theoretical lower bound, as unicast connections with shared sources or destinations must be placed in separate groups. Notably, the maximum clique algorithm approaches this theoretical limit, particularly in low-density networks.

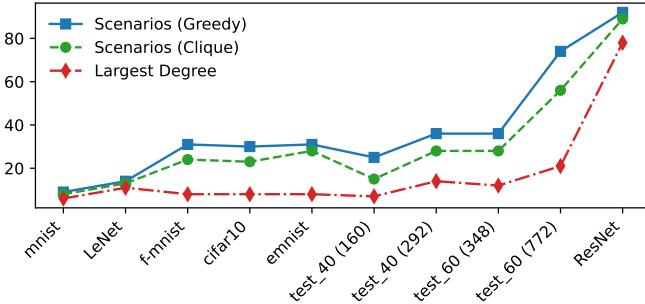


Fig. 5: Scenario scaling for different algorithms.

## V. CONCLUSIONS

This work proposed a scenario-aware control methodology for the segmented ladder bus, a dynamic interconnect designed

for large-scale neuromorphic systems. By leveraging compile-time traffic analysis, our control plane design minimizes the number of switching scenarios, reducing control memory and hardware overhead. Our FPGA implementation and simulation analysis show that the control plane remains lightweight, using under 10% of the total network resources. Scalability analysis further indicates that control complexity grows more slowly than network connectivity. Overall, our scenario-aware control plane design offers an efficient and scalable solution for neuromorphic interconnects.

## ACKNOWLEDGMENT

This work is supported by US DOE Award DE-SC0022014 and the US NSF Award CCF-1942697.

## REFERENCES

- [1] D. Kudithipudi, C. Schuman, C. M. Vineyard, T. Pandit, C. Merkel, R. Kubendran, J. B. Aimone, G. Orchard, C. Mayr, R. Benosman *et al.*, “Neuromorphic computing at scale,” *Nature*, 2025.
- [2] M. Yao, O. Richter, G. Zhao, N. Qiao, Y. Xing, D. Wang, T. Hu, W. Fang, T. Demirci, M. De Marchi *et al.*, “Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip,” *Nature Communications*, vol. 15, no. 1, p. 4464, 2024.
- [3] M. V. DeBole, B. Taba, A. Amir *et al.*, “TrueNorth: Accelerating from zero to 64 million neurons in 10 years,” *Computer*, 2019.
- [4] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *Ieee Micro*, 2018.
- [5] C. Effiong, G. Sassatelli, and A. Gamatié, “Combined distributed shared-buffered and diagonally-linked mesh topology for high-performance interconnect,” *Micromachines*, vol. 13, no. 12, p. 2246, 2022.
- [6] D. Xu, Y. Ouyang, W. Zhou, and H. Liang, “Improving power and performance of on-chip network through virtual channel sharing and power gating,” *Integration*, vol. 93, p. 102059, 2023.
- [7] A. Balaji, Y. Wu, A. Das, F. Catthoor, and S. Schaafsmma, “Exploration of segmented bus as scalable global interconnect for neuromorphic computing,” in *Great Lakes Symposium on VLSI (GLSVLSI)*, 2019.
- [8] A. Balaji, P. K. Huynh *et al.*, “Neusb: A scalable interconnect architecture for spiking neuromorphic hardware,” *IEEE Transactions on Emerging Topics in Computing (TETC)*, 2023.
- [9] P. K. Huynh, I. Mustafazade, F. Catthoor, N. Kandasamy, and A. Das, “A scalable dynamic segmented bus interconnect for neuromorphic architectures,” *IEEE Embedded Systems Letters*, 2024.
- [10] F. Catthoor, T. Basten *et al.*, *System-scenario-based design principles and applications*. Springer, 2020.
- [11] I. Mustafazade, N. Kandasamy, and A. Das, “Clustering and allocation of spiking neural networks on crossbar-based neuromorphic architecture,” in *Proceedings of the 21st ACM International Conference on Computing Frontiers*, 2024.
- [12] P. K. Huynh, F. Catthoor, and A. Das, “Mapping and scheduling spiking neural networks on segmented ladder bus architectures,” *Journal of Systems Architecture*, vol. 169, p. 103590, 2025.
- [13] P. Raghavan, A. Lambrechts, M. Jayapala, F. Catthoor, and D. Verkest, “Distributed loop controller for multithreading in unithreaded ilp architectures,” *IEEE Transactions on Computers*, 2008.
- [14] H. Johnston, “Cliques of a graph-variations on the bron-kerbosch algorithm,” *International Journal of Computer & Information Sciences*, vol. 5, no. 3, pp. 209–238, 1976.
- [15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, 2002.
- [16] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
- [17] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, “Emnist: Extending mnist to handwritten letters,” in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.
- [18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.