

Exploring High-Level Neural Networks Architectures for Efficient Spiking Neural Networks Implementation

Riadul Islam, *Senior Member, IEEE*, Patrick Majurski, Jun Kwon, Sri Ranga Sai Krishna Tummala
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, Maryland, USA
{riaduli, majurski, junkyuk1, stummal1}@umbc.edu

Abstract—The microprocessor industry faces several challenges: total power consumption, processor speed, and increasing chip cost. It is visible that the processor speed in the last decade has not improved and saturated around 2 GHz to 5 GHz. Researchers believe that brain-inspired computing has great potential to resolve these problems. The spiking neural network (SNN) exhibits excellent power performance compared to the conventional design. However, we identified several key challenges to implementing large-scale neural networks (NNs) on silicon, such as nonexistent automated tools and requirements of many-domain expertise, and existing algorithms can not partition and place large-scale SNN computation efficiently on the hardware. In this research, we propose to develop an automated tool flow that can convert any NN to an SNN. In this process, we will develop a novel graph-partitioning algorithm and place SNN on a network-on-chip (NoC) to enable future energy-efficient and high-performance computing.

Index Terms—Artificial neural network (ANN), spiking neural network (SNN), convolutional neural network (CNN), ANN-to-SNN conversion.

I. INTRODUCTION

Spiking Neural Networks (SNNs) are the next generation of artificial neural networks (ANNs) inspired by biological organisms. These networks can be efficiently implemented in Neuromorphic platforms consisting of multiple processing cores where a fixed number of neurons are mapped to the cores. The communication among the neurons in synapses is facilitated by the network-on-chip (NoC) architecture, a de facto design choice for enabling communication in a multicore system. The conventional SNN uses artificial spiking neurons and crossbars similar to the cache memory architecture [1]. The neuron fires (i.e., produces spikes) as soon as it exceeds its action potential threshold, and crossbars store the synaptic weights [2]. The computational efficiency in terms of execution time and power consumption of the SNN-based neuromorphic platform depends on the optimal mapping of those neurons to the cores with minimum communications delay. However, the electrical limitation of fan-in and fan-out constrained the number of synapses per neuron. As a result, NoC architectures are used

to integrate multiple crossbars. However, existing mapping strategies for mapping neurons to the cores in a multicore system are agnostic of the underlying NoC architecture and do not ensure minimum communication latency. Besides, we identified several critical challenges in implementing large-scale SNNs in real-hardware platforms, and these are: (i) Existing approaches have no real guidelines for high-level architecture building to hardware implementation; (ii) The nonexistent automated tools and requirements of many-domain expertise; (iii) Existing neuron clustering approaches could not handle more than a few thousand neurons in an SNN. This research resolves the above issues by proposing a novel graph-partitioning algorithm and placing SNN models on a NoC architecture with a generic tool flow.

In this paper, we resolve the major bottleneck of existing graph-partitioning algorithms [1], [3], [4], where the number of vertices limited to below 10 k. The proposed greedy graph-partitioning algorithm can handle graphs with more than 100 k vertices and reduces a significant amount of communication when placed in the crossbar hardware. In particular, the specific contributions of this work are:

- We propose an automated tool to convert any ANN and convolutional neural network (CNN) into an SNN for energy-efficient computing systems.
- We propose a novel graph-partitioning algorithm to implement large SNNs.
- We benchmark different deep NN (DNN) architectures and integrate various applications to show the efficiency of the proposed methodology.

II. BACKGROUND

In the past decade, researchers paying great attention to SNN. The primary reason is the energy efficient operation [5]. Unlike conventional low-power techniques [6]–[9], SNN models inherently respond to event-based data and suitable for address event representation-based computation [10]. The CARLsim is a C++ library that is commonly used to train and simulate large biologically detailed neural networks (NNs) [5]. The simulator supports the concurrent use of multiple CPUs and GPUs for heterogeneous computing platforms. Another

This work was supported in part by Federal Work-Study (FWS) award, National Science Foundation (NSF) award number: 2138253, Rezonent Inc. award number: CORP0061, and the UMBC Startup grant.

interesting automation tool uses a pre-trained ANN to construct an SNN, named SNN tool Box (SNN-TB) [11]. This tool's significant advantage is that one can extract the SNN model and deploy to the existing SNN simulator [12]. Hence, we utilized this tool for graph extraction.

Graph-partitioning is an essential technique in the electronic design automation flow, where a heuristic based on Kernighan-Lin (KL) algorithm [13] for graph bipartition is trivial. The major bottleneck of existing graph-partitioning algorithms [1], [3], [4] is that the number of vertices is limited to below 10 k. Hence, this paper resolves this issue by proposing an SNN graph-partitioning algorithm (SNN-GPA) that can handle graphs with more than 100 k vertices and reduces a significant amount of communication when placed in the crossbar hardware.

III. PROPOSED SPIKING NEURAL NETWORK IMPLEMENTATION TOOL FLOW

A. Spiking Neural Network Characterization

In this research, we propose a comprehensive platform for high-level neural architecture exploration to characterize and implementation of SNN accurately. First of all, we train a NN model using a conventional software library (i.e., Keras [14]), and it is implemented in the TensorFlow platform [15]. The proposed tool flow supports conventional dense (fully connected), 2D convolutional, average pooling, flattening, dropout, and batch normalization Keras layers. Then we convert the trained NN model using an open-source SNN converter [11] into an SNN model. The SNN converter translates the ANN/CNN into an SNN in two steps. First, the ANN is parsed into an intermediate architecture where dropout and batch normalization layers are either removed or incorporated into adjacent layers. The weights for each layer are normalized at this stage. Second, the parsed CNN is converted into an SNN model. The SNN converter supports several simulator backends [12], [16], but currently, the built-in INI backend is used with a temporal mean rate approximation.

Unlike conventional methodologies, we validate the accuracy of the SNN model using the test data. Then, we extract the spike events of each neuron and the classification accuracy. The constant current and the Poisson spike trains can be utilized as input current. Then, the proposed methodology uses the SNN architecture and spike event data to translate the ANN connectivity between layers into equivalent synaptic connections. The overall tool flow is shown in Figure 1. **We build an SNN graph considering the presynaptic neuron fire rate and the weight of the connection.**

B. Graph Partitioning

Once an SNN is trained, we extract a connected graph. Then, depending on the synaptic weight, the proposed graph partitioning algorithm clusters neurons considering pair-wise SNN-layers. The algorithm is based on the existing KL algorithm [13]. It considers the intra-communication (i.e., neurons reside in the same cluster) weights (*IntraW*) and

inter-communication (i.e., neurons reside in different clusters) weights (*InterW*) of the synapses.

Algorithm 1 SNN Graph partitioning algorithm

```

1: Input:  $G(V, E)$ , Graph with  $V$  vertices and  $E$  edges;  $K$ , # of Clusters;  $n$ , # of layers;
2: Output:  $P$  partitions;
3:
4:  $K' = K/(n - 1)$   $\triangleright$  Compute the total number of clusters in a pair-of layer.
5:  $G' = \{G'_1, G'_2, \dots, G'_n\} = \text{RandGraph}(G, n)$   $\triangleright$  Create a random partition considering consecutive layers
6: for all  $G'_i \in G'$  do :
7:    $C = \{C_1, C_2, \dots, C_k\} = \text{RandCut}(G'_i, K')$   $\triangleright$  Create random clusters from each initial partitions.
8:   for all  $\{C_i, C_j\} \in C$  do :
9:      $\{C_i, C_j\} = \text{KLmethod}(C_i, C_j)$   $\triangleright$  Apply the KL-based method to improve intra-cluster weights and reduce inter-cluster weights.
10:  end for
11: end for
12:  $P = G'$   $\triangleright$  Assigned clusters to partitions list.
13: return  $P$ 

```

The proposed SNN graph-partitioning algorithm (SNN-GPA) is shown in Algorithm 1. The SNN-GPA takes an SNN as a graph (G), the number of clusters (K), and the number of layers (n) as inputs and returns partitions or a list of subgraphs. Here, the number of clusters depends on the constraint of the maximum number of neurons per cluster. The SNN-GPA computes the number of clusters in a pair-of layers and creates a set of random graphs/partitions considering consecutive layers in Line 4 and Line 5, respectively. Then, it iteratively converts each partitions into random clusters and applies *KLmethod*() to maximize *IntraW* and minimize *InterW* using Line 6 to Line 11. Then, the updated clusters are assigned to output partitions Line 12, and SNN-GPA returns the partitions in Line 13. The optimal partition with trained SNN is then used to map on hardware grids for deployment.

IV. RESULTS AND DISCUSSION

A. Experimental Setup

For initial results, we performed analysis on an Intel 32-cores Intel Xeon Gold processor with 64 GB RAM, NVIDIA Quadro P4000 GPU running Ubuntu 18.04. We used Python programming language to build the tool flow. We used both synthetic and realistic networks. We used the 3-layer synthetic feedforward network with 4000 neurons and 3.75M synapses (i.e., synthetic_4k network). For realistic networks, we used CNN_mnist [1], LeNet_mnist [17], Zambrano_mnist [18], Rueckauer_Cifar10 [11], LeNet_cifar10 [19], AlexNet_mnist [20], multilayer perceptron for mnist (MLP_mnist), a CNN for DogsVsDogs [21], a CNN for Fruits360 [22], and AlexNet_CatsVsDogs [20] for benchmarking. For analysis, we used mnist handwritten digit [17], Cifar10, and CatsVsDogs [21] datasets.

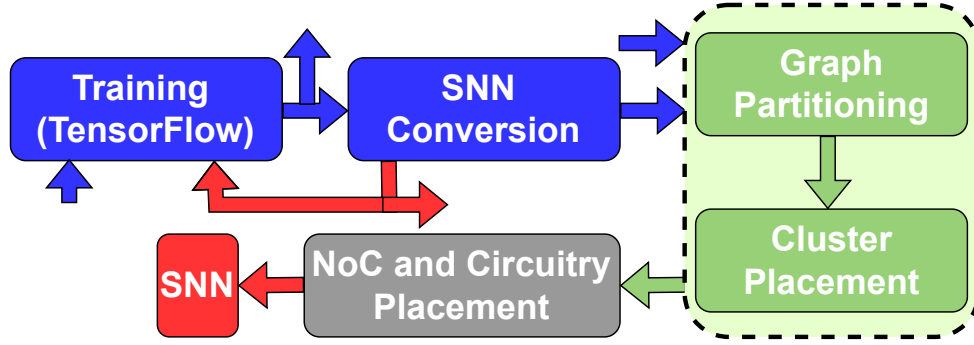


Fig. 1: The proposed tool flow trains an ANN model using TensorFlow, converts the trained model to SNN, and then applies the proposed graph partitioning algorithm to cluster and place it on an NoC grid.

B. Implementation Results

We implemented each network using TensorFlow and trained using the Keras library. To convert a CNN or ANN into an SNN, we used a modified version of SNN-TB [11]. It is worth mentioning that the normalization of CNN weights is critical for accurate SNN models. Unfortunately, the SNN-TB implementation requires the layer activations for the entire normalization dataset to be loaded into GPU memory at once. As a result, large models with large datasets can't be normalized on most GPUs. As a result, we created a normalization workaround that generates identical results but shifts the space constraint to system memory.

To verify the efficiency of the ANN-to-SNN conversion, we used Pearson correlation coefficients considering ANN activations and SNN spike rates. Figure 2 shows the CNN_mnist network's correlated coefficients for each layer trained for the mnist dataset, averaged over all the batches.

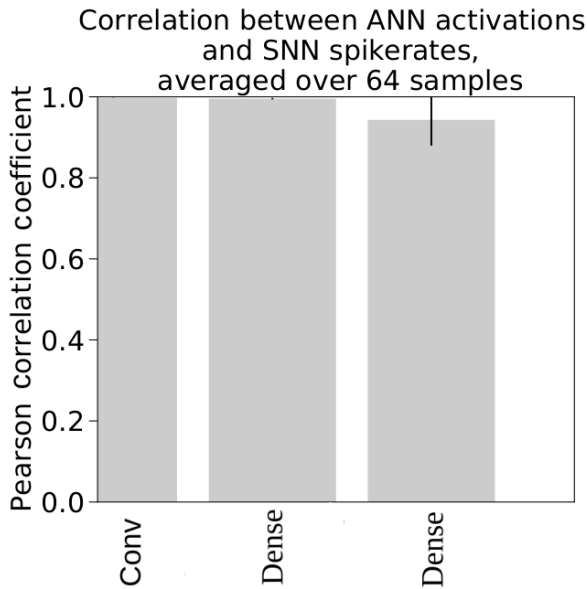


Fig. 2: The Pearson correlation coefficients confirm the ANN-layers activations and SNN-layers spike rates for CNN_mnist.

Besides, we traced the evolution of the classification error during training for different values of the communication periods, as shown in Figure 3. Simulation time is taken in steps of 1 ms. The scatter points in green represent the top-1 errors over the time and the points in blue represent the top-5 errors over the time. The shaded area represents the standard deviation of the classification errors by SNN and ANN, respectively.

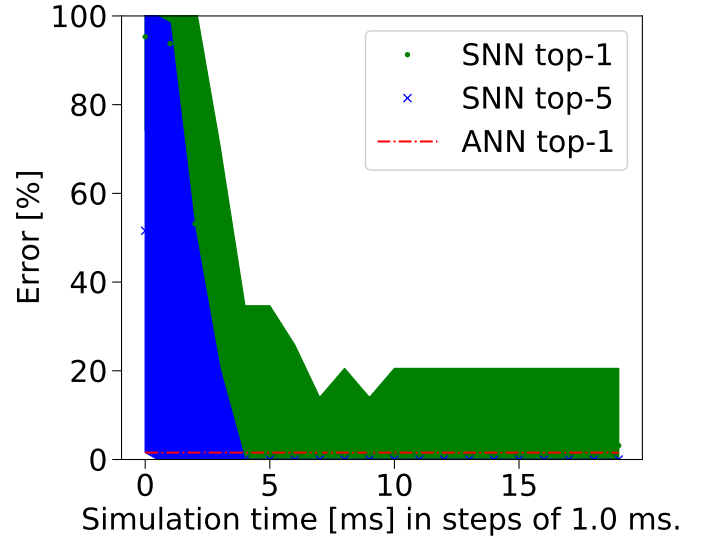


Fig. 3: As expected, the classification error of the SNN model reduces with the increase of simulation time, while the ANN remains nearly constant, as we provided a trained ANN model for this analysis.

The realistic networks ANN and SNN accuracy comparison is shown in Table I. The networks uses 173.8 M synapses and 0.39 M neurons on average, respectively. The average number of spike count and simulation time (S_T) are 8476.14 M and 320 s, respectively. The proposed methodology can efficiently converts ANN architectures to SNN with only 2.65% average error penalty.

To analyze the efficiency of the proposed graph-partitioning algorithm, we consider the *IntraW* and *InterW* synaptic

TABLE I: The average number of synapses and neurons are 173.8 M and 0.39 M, respectively; The proposed tool flow efficiently converts ANN to SNN considering realistic benchmarks with only 2.65% average error penalty.

Application	# of synapses (M)	# of neurons (M)	# of spikes (M)	S_T (s)	SNN acc. (%)	ANN acc. (%)
CNN_mnist [1]	1.61	0.01	79.33	200	97.947	98.46
LeNet_mnist [17]	0.29	0.007	93.72	200	98.037	98.88
Zambrano_mnist [18]	1.42	0.01	125.10	200	99.25	99.36
Rueckauer_Cifar10 [11]	2.50	0.11	7786.46	1000	79.43	81.25
LeNet_Cifar10 [19]	0.66	0.01	752.26	200	53.596	60.64
AlexNet_mnist [20]	923.45	0.79	9550.44	500	97.24	98.54
MLP_mnist	0.20	0.001	30.27	200	97.33	97.60
CNN_CatsVsDogs [21]	522.82	2.03	9290.56	50	91.60	93.88
CNN_Fruits360 [22]	96.09	0.40	48092.77	600	89.5	96.71
AlexNet_CatsVsDogs [20]	165.01	0.53	8960.47	50	77.7	82.78
Average	173.80	0.39	8476.14	320	88.16	90.81

weights. For this analysis, we used both synthetic and realistic networks considering mnist dataset. In addition, we used an SNN architecture for standard edge detection graph_edgedet. Using the Zambrano_mnist network, the proposed graph-partitioning algorithm can reduce 6.65% and 99.86% inter-communication and intra-communication weights compared to a baseline model, respectively. Overall, the proposed SNN-GPA reduce 14.22% and 87.58% inter-communication and intra-communication weights compared to a baseline model.

Once we create the partitions, then the proposed tool flow can place those neurons on a dedicated NoC grid. We designed a 2D mesh NoC architecture considering a 2 nm grid length and using a Cartesian coordinate system. However, this is a design choice and can vary from 10 μm to hundreds of micrometers [23]. Figure 4 exhibits a representative diagram when proposed tool flow places Zambrano_mnist in a 120×120 mm chip.

V. CONCLUSION

This paper proposed a comprehensive tool flow to explore high-level NN architectures that can efficiently implement and explore SNN models. The tool flow uses Python Keras libraries, SNN-TB, and our proposed SNN-GPA algorithm partitions and place SNN on an NoC architecture. The proposed methodology efficiently converts ANN architectures to SNN with only a 2.65% average error penalty. In addition, the proposed SNN-GPA reduces 7.28% and 83.18% inter-communication and intra-communication weights compared to a baseline model.

REFERENCES

- [1] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2020.
- [2] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

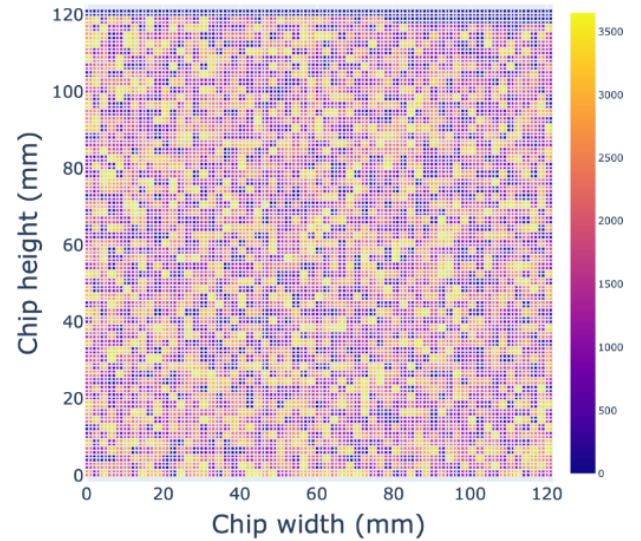


Fig. 4: A representative diagram when proposed tool flow places Zambrano_mnist in a 120×120 mm chip.

- [3] A. Das, Y. Wu, K. Huynh, F. Dell'Anna, F. Catthoor, and S. Schaafsma, "Mapping of local and global synapses on spiking neuromorphic hardware," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1217–1222.
- [4] G. Karypis and V. Kumar, "METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.
- [5] T.-S. Chou, H. J. Kashyap, J. Xing, S. Listopad, E. L. Rounds, M. Beyeler, N. D. Dutt, and J. L. Krichmar, "CARLsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters," *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.
- [6] R. Islam and M. R. Guthaus, "CMCS: Current-mode clock synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1054–1062, 2017.
- [7] R. Islam, B. Saha, and I. Bezzam, "Resonant energy recycling sram architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 4, pp. 1383–1387, 2021.
- [8] R. Islam, H. Fahmy, P.-Y. Lin, and M. R. Guthaus, "Differential current-mode clock distribution," in *International Midwest Symposium*

TABLE II: The proposed SNN-GPA uses both synthetic and realistic networks; When compared with a base graph-partitioning algorithm, the proposed approach reduces on average 14.22% inter synaptic communication and 87.58% intra synaptic communication.

Benchmark	# of nodes	# of clusters	\$ of edges	InterW (base)	IntraW (base)	InterW (our)	IntraW (our)	Inter (%)	Intra (%)	Runtime (s)
synthetic_4k	4000	250	3750000	536702.65	1877.17	535628.61	2951.21	0.20	36.39	373.16
MLP_mnist	1050	66	203264	918.05	12.05	465.10	465.05	49.34	97.41	3.06
graph_edgedet	4377	274	54876	5332848.19	4421.91	4322871.32	1014398.78	18.94	99.56	53.51
LeNet_mnist	6598	413	286120	5236.54	9.59	4967.05	279.08	5.15	96.56	235.80
LeNet_mnist_padded	9118	570	422824	5885.53	10.45	5588.75	307.24	5.04	96.60	371.46
Zambrano_mnist	14554	910	1422848	53811.83	37.37	50235.91	3613.29	6.65	98.97	1113.34
Average improvement								14.22	87.58	

on Circuits and Systems (MWSCAS), 2015, pp. 1–4.

- [9] R. Islam and M. R. Guthaus, “HCDN: Hybrid-mode clock distribution networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 251–262, 2019.
- [10] Q. Zhou and X. Li, “A bio-inspired hierarchical spiking neural network with reward-modulated stdp learning rule for aer object recognition,” *IEEE Sensors Journal*, vol. 22, no. 16, pp. 16 323–16 338, 2022.
- [11] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification,” *Frontiers in Neuroscience*, vol. 11, 2017. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fnins.2017.00682>
- [12] M. Stimberg, R. Brette, and D. F. Goodman, “Brian 2, an intuitive and efficient neural simulator,” *eLife*, vol. 8, p. e47314, Aug. 2019.
- [13] B. W. Kernighan and S. Lin, “An efficient heuristic procedure for partitioning graphs,” *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.
- [14] F. Chollet *et al.* (2015) Keras. [Online]. Available: <https://github.com/fchollet/keras>
- [15] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [16] M.-O. Gewaltig and M. Diesmann, “Nest (neural simulation tool),” *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.
- [17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [18] D. Zambrano and S. M. Bohte, “Fast and efficient asynchronous neural computation with adapting spiking neural networks,” *arXiv preprint arXiv:1609.02053*, 2016.
- [19] X. Zhang, “The alexnet, lenet-5 and vgg net applied to cifar-10,” in *International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, 2021, pp. 414–419.
- [20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, p. 84–90, may 2017. [Online]. Available: <https://doi.org/10.1145/3065386>
- [21] Kaggle, “A dataset with 90380 images of 131 fruits and vegetables,” <https://www.kaggle.com/datasets/moltean/fruits>, accessed September 30, 2022.
- [22] M. Oltean, “Create an algorithm to distinguish dogs from cats,” <https://www.kaggle.com/competitions/dogs-vs-cats/data>, accessed September 30, 2022.
- [23] G. Fayez and E. Haytham, “Networks-on-chips: Theory and practice,” https://users.auth.gr/ksiop/publications/crc_2009_noc.pdf/, Last accessed November 23, 2022.