An FPGA-based neuromorphic vision system accelerator

Teymoor Ali[a], James Rainey[a], Sook Yen Lau[a], Elena Gheorghiu[b], Patrick Maier[c], Kofi Appiah[d], and Deepayan Bhowmik[a]

[a]School of Computing, Newcastle University, Newcastle upon Tyne, UK
[b]Division of Psychology, University of Stirling, Stirling, UK
[c]Division of Computing Science and Mathematics, University of Stirling, Stirling, UK
[d]Department of Computer Science, University of York, York, UK

ABSTRACT

Rapid reaction to a specific event is a critical feature for an embedded computer vision system to ensure reliable and secure interaction with the environment in resource-limited real-time applications. This requires high-level scene understanding with ultra-fast processing capabilities and the ability to operate at extremely low power. Existing vision systems, which rely on traditional computation techniques, including deep learning-based approaches, are limited by the compute capabilities due to large power dissipation and slow off-chip memory access. These challenges are evident in environments with constrained power, bandwidth and hardware resources, such as in the applications of drones and robot navigation in expansive areas.

A new NEuromorphic Vision System (NEVIS) is proposed to address the limitations of existing computer vision systems for many resource-limited real-time applications. NEVIS mimics the efficiency of the human visual system by encoding visual signals into spikes, which are processed by neurons with synaptic connections. The potential of NEVIS is explored through an FPGA-based accelerator implementation on a Xilinx Kria board that achieved 40× speed up compared to a Raspberry Pi 4B CPU. This work informs the future potential of NEVIS in embedded computer vision system development.

Keywords: Neuromorphic computing, Vision system, Accelerator, FPGA.

1. INTRODUCTION

The advancement of computer vision algorithms, especially with the rise of deep learning-based algorithms, enables their use in everyday life. However, the data-intensive nature of computer vision algorithms dictates the need for the support of powerful computing hardware such as multi-core CPUs, GPUs, or specialised vision accelerators, which are often inefficient both in terms of speed and power dissipation. As an alternative to the general computing paradigm (typically von Neuman-like architecture), there has been considerable interest among researchers and industry alike in developing neuromorphic hardware and algorithms that replicate the brain's architecture and function for efficient low-energy cognitive computing. For example, prominent platforms such as SpiNNaker,[1] Intel Loihi,[2] Braindrop[3] and IBM TrueNorth[4] have been developed to address current bottleneck with the von-Neumann memory access. These platforms accelerate the neural network simulations by incorporating the brain-inspired model consisting neurons and synapses, which both process and store values in parallel across extensive network applications.

The above-mentioned neuromorphic hardware are silicon or complementary metal-oxide-semiconductor (CMOS) materials. Various other materials are also being explored for neuromorphic applications such as phase-change, ferroelectric, topological insulators or channel-doped biomembranes.[5–7] One of the widely studied approaches is using memristors,[8,9] which possess resistive memory for collocating processing and memory tasks, to mimic the functions of biological synapses and neurons. It is worth noting that each device and material used in neuromorphic hardware has unique operating characteristic such as speed performance, energy consumption and biological resemblance.

Send correspondence to Deepayan Bhowmik: deepayan.bhowmik@newcastle.ac.uk

The neurons communicate through spike trains, which can be modelled by using Spiking Neural Network (SNN) that rely on analog computation and event-driven processing, where neurons only work when there are spikes to process.[7] As a result, the network with spatio-temporal information encoding capabilities enables highly efficient computation, which is ideal for processing real-time tasks in portable hardware implementations. Synapses, which form the connections between neurons, have associated weight value (synaptic strength) that changes over time in response to network activity. These synaptic weights can be modulated by an unsupervised learning mechanism in SNN, commonly known as Spike-timing-dependent plasticity (STDP), which is induced by the relative spike timings from the pre- and post-synaptic neurons.

Numerous studies have explored various properties of SNN[10] such as neuron models,[11,12] signal encoders,[13–15] learning algorithms,[16,17] and network architectures[18,19] and demonstrated that both information transmissions between neurons and internal neuron processing are biologically plausible. However, the increasing complexity of network models and computation requirements in SNN inference has created a need for customised hardware accelerators to balance the hardware, power consumption, and acceleration performance.

While efforts were made for general neuromorphic computing, only a few approaches are available that propose mimicking human vision as an alternative to current vision system accelerators implemented on CPU or GPU-like processor architectures. Additionally, general neuromorphic processing architectures do not consider visual information processing flow, which is crucial for efficient vision computing; are often expensive and, therefore, not readily suitable for many usable vision applications in today's world. In addressing such challenges, we propose a new NEuromorphic Vision System (NEVIS). This paper describes the initial development of NEVIS and its implementation on hardware. The main contributions of this work are:

- A computational model mimicking early stages of human vision by using oriented convolutional filters and a spiking neural network.
- An FPGA-based implementation of the computational model demonstrating the new NEVIS system.
- An experimental validation of the NEVIS system on the popular MNIST digit classification dataset with 40× speed up on the custom accelerator over regular CPUs.

2. SNN APPLICATIONS ON HARDWARE

SNN implementations on different hardware, such as Central Processing Units (CPUs) or Graphics Processing Units (GPUs), have demonstrated slow performance with high power overhead due to the limited memory bandwidth.[20] It is to be noted that in SNNs, neural processing and synaptic memory are closely integrated, which results in a highly-distributed computing model that cannot be effectively implemented on CPUs and GPUs. Research works have also been focused on developing custom Application-Specific Integrated Circuits (ASICs) to optimise the performance and energy efficiency of SNN applications. However, ASICs lack the flexibility to adapt to the rapid evolution of SNN models due to its high non-recurring engineering cost and lengthy process of design, verification and fabrication for ASIC chips.[21,22] Therefore, FPGA has emerged as a potential platform for SNN acceleration, offering advantages including logic reconfigurability, high power efficiency, low latency, and support for parallel computing as well as bit-level operations, particularly in visual processing applications.

A few related research were proposed in the literature: Caron et al.[23] simplified a hardware-SNN from the ODLM algorithms and implemented it on an FPGA for pattern matching; Yasukawa et al.[24] employed analog MOS-based resistive networks for high-speed spatial filtering and an FPGA to process SIFT algorithms for features tracking; Ju et al.[25] implemented a deep-SNN for fast MNIST classification by using 79,322 neurons on a single FPGA, demonstrating that the application consumes less than 52% of the available resources on the leading neuromorphic platforms. The studies indicated that most of the existing neuromorphic architectures are generic, featuring many-core neuron processing units with programmable communication intersections, and directly converting models from convolution neural network (CNNs) to spike-based neural network (SNNs). However, the generic neuromorphic processor might not be effective in optimising vision applications when compared to a domain-specific vision processor, which is designed based on the human visual system that excels in highly power-efficient event-based signal processing.

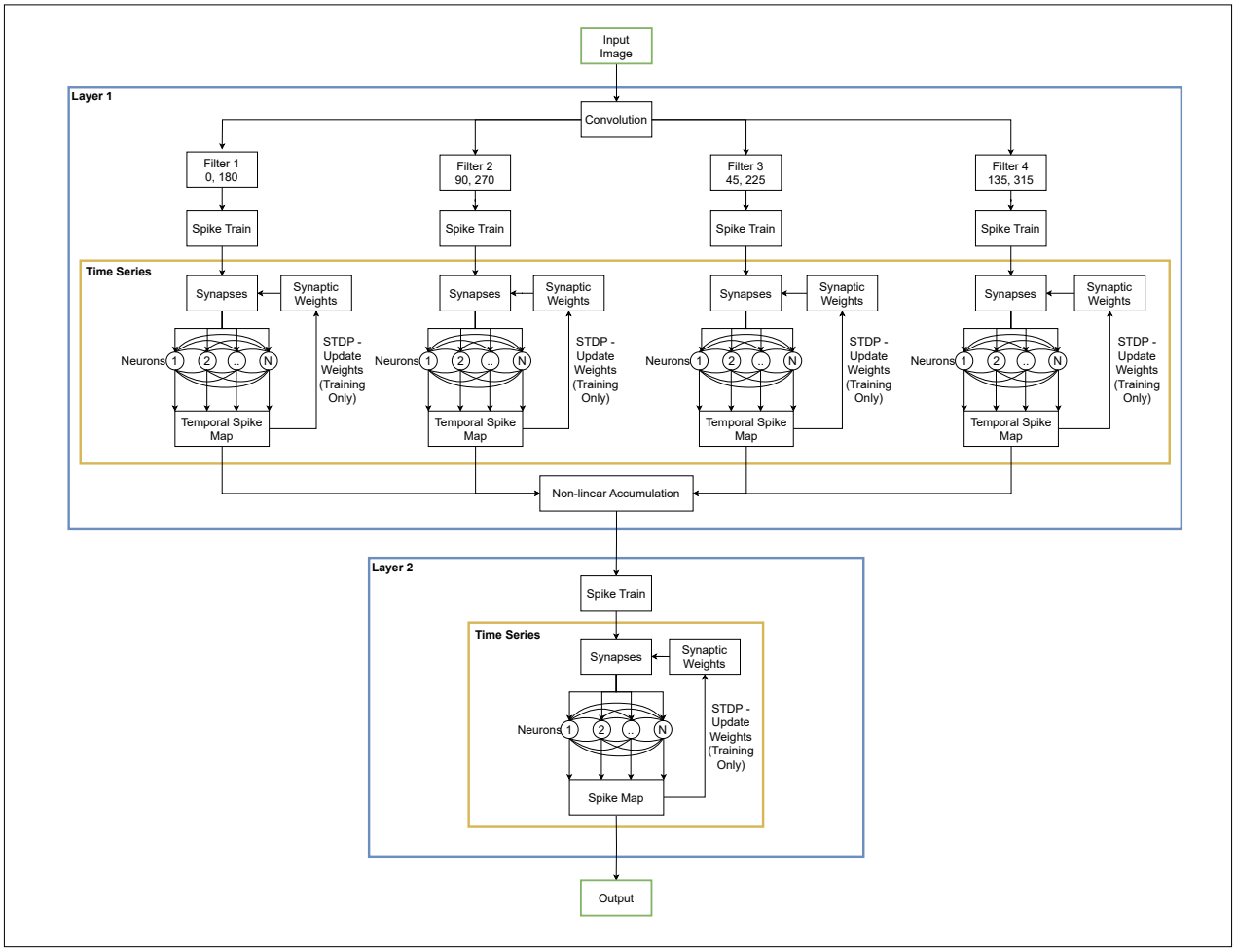

Figure 1. Proposed network architecture consisting of 2 layers

3. METHODOLOGY

3.1 Computation modelling

The overall initial architecture of the NEVIS model is shown in Figure 1. The proposed architecture takes into consideration the MNIST dataset, which has ten digits to classify. Our current model consists of two layers: the first layer is made of multiple convolution filters which extract orientations from the input and a set of neurons for each filter. The second layer is a set of ten neurons, one for each class (targeting ten digits in MNIST dataset), which produces the accumulated output. We compared two types of operators for the convolutional filters: a basic operator $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, and a Prewitt operator $\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$. The neurons are implemented using the Izhikevich neuron model,[11] an efficient neuron model that simulates the behaviour of neurons in the brain.

In the first layer for each filter a spike train is calculated, this encodes the frequency of the input as spikes that are passed into the neurons. These spikes are propagated to the neurons through the synapses as output spikes, which have weights that influence the size of the spikes. At the start of the training phase the synapses weights are randomly initialised from a uniform distribution. Throughout the training, STDP is used to update the weights and learn features from the data. For testing the trained weights are loaded and STDP is disabled.

STDP updates the weights based on the timing of input spikes relative to output spikes. An input spike

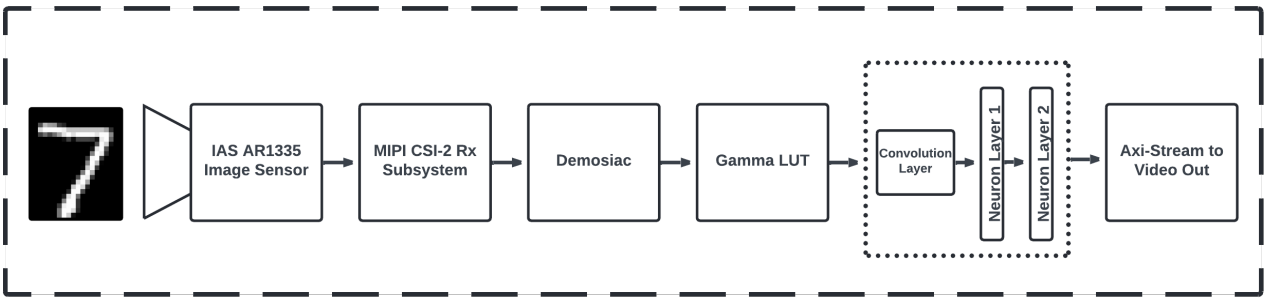

Figure 2. Implementation of Spiking Neural Network on FPGA.

occurring immediately before an output spike will have a supporting effect and will strengthen the spike, thus the weighting of that neuron will be increased. Conversely, an input spike occurring immediately after an output spike will have an inhibitory effect and will weaken the spike, resulting in the weighting being reduced.

Between the neurons there are synapse connections which allow for lateral inhibition, in which the neurons that are spiking inhibit the firing of neighbouring neurons. The output from each of the sets of neurons is a temporal spike map which contains the spikes produced by the neurons, these are accumulated to form a single representation that is passed to the second layer. The second layer consists of a single set of neurons which has the same number of neurons as classes in the training data.

3.1.1 Training

The network implements a biologically plausible unsupervised competitive learning approach, similar to self-organising maps, that learns representations of digits by adapting the input weights of excitatory neurons using STDP. The network by Diehl and Cook[26] learns representations of 28x28 pixel images. Instead, our model learns representations based on 14x14x2 filters, oriented horizontally and vertically. We trained our model on a subset of 6000 MNIST images. In order to use the network for classification, training is followed by a labelling step. With input weights frozen, the network runs over 1000 MNIST images, recording the spiking frequency of each neuron on each input image. This defines probability distributions over the 10 classes of digits, for each of the 100 excitatory neurons. These distributions indicate how accurate the learned representations are: the higher the probability of the mode of a neuron's distribution, the more accurately it represents a digit.

3.1.2 Prediction

To classify unknown inputs the network is run over an input image. The weights from the training are loaded into the synapses and the image is passed through the network. After the first layer, the outputs are accumulated and passed to the second layer. The predicted class is determined by the output of the second layer of neurons, with each neuron relating to a class.

3.2 Hardware system implementation

In realising our computational model as a usable portable vision system, we chose a lightweight FPGA vision kit Xilinx Kria *. The model is also implemented on a Raspberry Pi 4B (embedded CPU) for comparison purposes.

3.2.1 FPGA implementation

The proposed implementation leverages the inherent low-latency and parallelism of FPGA architecture. The design ensures the high-throughput demands of an SNN, providing real-time processing of spikes while maintaining scalability for larger networks. The proposed implementation shown in Figure 2 uses the configuration with the best accuracy shown in Table 1. The proposed SNN design was developed using simulation software whilst the image capture and processing pipeline (*e.g.*, De-mosaicing, Gamma Correction) was executed on hardware

*https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html

to obtain image pre-processing runtime results (excluding the SNN processing). In addition, only inference was implemented, as training was not included due to the limited on-chip resources.

The hardware implementation of the SNN is structured into two main modules: the convolution and STDP within the first layer, and the second layer, each carefully designed to leverage the parallel processing capabilities of the FPGA. The convolution layer performs convolution operations on input spikes using 16 parallel 3×3 filters. Instead of relying on sequential loops, as typically seen in software implementations like Python, the hardware design parallelises the filtering process. Each filter operates concurrently, processing different regions of the input spike map. The input spike data is managed through a FIFO (First In, First Out) buffer, allowing the filters to process spikes without stalling or requiring sequential access. The output of the convolution layer is a new set of spikes, which are pipelined to the next module for further processing.

In the first hidden layer, 36 neurons are instantiated separately, allowing each neuron to operate independently in parallel. Each neuron accumulates membrane potentials based on the spike inputs from the convolution layer and performs multiplication with the corresponding weights. The neurons in this layer use a thresholding mechanism to trigger spike generation when their accumulated membrane potential exceeds a pre-defined threshold. The weights for each neuron are retrieved from DDR memory using a memory controller, which ensures that data is fetched without overwhelming the limited on-chip memory. Each neuron has its own potential accumulator, and the spike output is registered for further processing in the subsequent layer. The second hidden layer serves as the classification layer, comprising 10 neurons. Similar to the first hidden layer, each neuron operates in parallel, independently accumulating potentials based on inputs from the previous layer and retrieving corresponding weights from external DDR memory. The spike outputs are again stored in a FIFO buffer to prevent data congestion.

To handle weight storage, a dedicated DDR memory controller manages the retrieval and storage of neuron weights. This controller is optimised to minimise memory access delays by using burst reads and write-back caching techniques, ensuring that the system does not bottleneck during weight loading. Additionally, state machine logic ensures that multiple neurons can request weight data simultaneously without causing access conflicts, maximising the available memory bandwidth.

3.2.2 Raspberry Pi implementation

To provide a CPU-based comparison, the SNN is also implemented on a Raspberry Pi. For a better relative comparison, the clock speed of the Raspberry Pi is limited to 600Mhz.

The network is fully defined in Python, and as with the FPGA it is implemented in three parts, first the convolutions, then the two neuron layers. For each of the orientation filters the image is broken into patches and a convolution is defined using either a basic or prewitt kernel, this is then combined with a set of neurons which makes up the first layer. The output of this is passed to an accumulator which accumulates the neuron spikes into an input for the second layer. The second layer is defined as a set of ten neurons, one for each class. Unlike the FPGA implementation, the neuron layers are implemented using sequential loops, with each set of neurons stored in an array which is iterated over. Each image patch has to wait for the preceding patch to completed before being processed.

3.3 Power and Execution Time Measurement

FPGA power consumption is typically measured using tools such as Vivado Power Analysis [†]. After completing the synthesis and implementation stages of the FPGA design in Xilinx Vivado, the tool generates detailed power estimation reports. The report shows both static and dynamic power consumption. Dynamic power refers to the power consumed by the switching activity in the FPGA fabric, while static power is due to leakage currents. Raspberry Pi devices lack built-in hardware counters to measure power consumption, so external tools are needed to monitor energy usage. In this experiment, a wattmeter (wall socket) is used to accurately measure the power drawn by the Raspberry Pi during the algorithm execution.

[†] `https://docs.amd.com/r/2021.1-English/ug907-vivado-power-analysis-optimization/Vivado-Power-Analysis`

Table 1. Test results (on desktop cpu) using the basic Operator (top) and Prewitt Operator (bottom).

Orientations	Layer 1 Neurons	Total Neurons	Inference Time(ms)	Accuracy (%)
2 (HV)	16	42	86	61.7
2 (HV)	36	82	164	63.8
4 (HVDD)	16	74	82	53.2
4 (HVDD)	36	154	158	64.8
2 (HV)	16	42	32	66.7
2 (HV)	36	82	65	67.6
4 (HVDD)	16	74	89	**71.7**
4 (HVDD)	36	154	98	68.4

Table 2. FPGA (Kria) and CPU (RPi) Hardware Comparison Results.

Hardware	Frequency (Mhz)	SNN Kernel Runtime (ms)	Power Consumption (W)
Kria KV260	200	38	3.8
Raspberry Pi 4B	600	1542	5.1

The execution time for the FPGA is measured using simulation software, which accurately models the hardware's behaviour. In the case of the Raspberry Pi, Linux runtime libraries are used to measure execution time by measuring the duration of algorithm execution.

4. RESULTS AND DISCUSSION

In understanding the implication of the proposed NEVIS architecture, several experimental combinations were created, and the results are shown in Table 1 and hardware comparisons of the selected configuration are observed Table 2. The experimental parameters include two different convolutional filters *i.e.*, basic and Prewitt, capturing different orientations, namely horizontal (H), vertical (V) and two diagonals (D) of opposite orientations and number of neurons (in layer 1). The total number of neurons is calculated as: $Layer\ 1\ Neurons * orientations + Layer\ 2\ Neurons$ (in these experiments, layer 2 is always 10 neurons). The NEVIS python implementation was trained and tested on the MNIST dataset, a subset of 6000 images from the dataset was used for the experiments.

Our observation indicates that accuracy was generally improved by introducing more orientations and more neurons. However, these are indicative and not always true, as the results are sensitive to spike rate, neuron weight learning rate, and input dataset. For example, in this case, the MNIST dataset has more horizontal and vertical edges, and therefore, greater accuracy is easily reached using just two orientation convolution filters. Generally, Prewitt operators performed better as they can distinguish edges more efficiently. This has implications for learning neuron weights, leading to faster operations. Experiments using Prewitt operator had faster inference time than those using the basic operator while also achieving the highest accuracies.

In terms of hardware comparison shown in Table 2, FPGAs like the Kria KV260 outperform traditional CPUs such as the Raspberry Pi 4B by a huge magnitude, even though the CPU operates at a much higher clock speed. This is primarily due to the ability of FPGAs to exploit parallelism and pipelining, making them ideal for the highly parallel nature of neural network processing. While the Raspberry Pi is clocked at 600 MHz compared to the Kria's 100 MHz, the Kria FPGA is approximately 40.57× faster in runtime and 1.34× more power-efficient. Table 3 reports the hardware resources used for the FPGA implementation.

Discussions: FPGAs offer advantages in terms of parallelism by enabling the concurrent processing of multiple neurons. Unlike CPUs, which are bound by sequential instruction execution and limited threading capabilities, FPGAs can instantiate dedicated hardware blocks for each neuron. This allows neurons to compute membrane potentials, process spikes, and generate outputs simultaneously. The result is a substantial performance boost, as

FPGAs can efficiently manage many parallel computations in real-time. The ability to create custom hardware circuits tailored for neuron processing is a key reason why FPGAs outperform CPUs in SNN workloads.

FPGAs can stream images directly from sensors to the programmable logic with low latency, bypassing the need for CPU initialization and reducing processing overhead. This allows the FPGA to update neuron states and process spikes as soon as the image data becomes available without waiting for a full image frame to be buffered. Additionally, the FPGA can start processing as soon as it receives enough lines of image data to cover the filter's receptive field, enabling convolution to begin immediately.

In contrast, CPUs are constrained by the von Neumann architecture despite their higher clock speeds. This design relies on a shared memory bus for both data and instructions, creating a bottleneck known as the von Neumann bottleneck. This sequential nature limits the CPU's ability to handle the high level of parallelism required by Spiking Neural Networks (SNNs). While CPUs excel at executing sequential tasks, they struggle to efficiently scale the concurrent processing needed for the massively parallel operations characteristic of SNNs, as each operation must pass through the same memory interface, reducing performance. Graphics Processing Units (GPUs) can be an alternative approach, offering better parallelism, but also suffer from high initialisation times, impacting real-time performance.

5. CONCLUSION

This work presents a new Spiking Neural Network implementation that addresses the limitations of existing computer vision systems in resource-limited applications. Therefore, exploiting the inherent parallelism and efficiency of FPGA-based hardware, this approach overcomes the challenges posed by traditional CPUs, which struggle with the high computational demands and power constraints in edge computing environments. The Kria (FPGA) achieves a significant performance advantage, being 40.57× faster and 1.34× more power-efficient than Raspberry Pi 4B (a popular embedded CPU), despite the CPU's higher clock speed. This is primarily due to the parallel processing approach used in the FPGA, which is well-suited for SNNs that require neurons to be processed in parallel, more accurately replicating the biological processes that occur in the human visual cortex.

This work demonstrates the suitability of FPGAs for implementing SNN methods, highlighting their effectiveness in handling the complex, parallel computations required for these tasks. Showcasing the potential of FPGA hardware to process SNNs efficiently, this research lays a strong foundation for further exploration into replicating the neuro-topology of human vision on computer hardware. The ability to mimic the parallel nature of neural activity leads to advancements in neuromorphic computing, enabling more accurate and biologically inspired models of human vision to be realised in resource-constrained edge environments.

REFERENCES

[1] Furber, S. and Bogdan, P. A., [*SpiNNaker: A Spiking Neural Network Architecture*], Now, Boston, Delft, first ed. (2020).

[2] Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., et al., "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro Computer Society* **38**, 82–99 (2018).

[3] Neckar, A., Fok, S., Benjamin, B., Stewart, T., Oza, N., Voelker, A., Eliasmith, C., Manohar, R., and Boahen, K., "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE* **107**, 144–164 (2019).

[4] DeBole, M., Taba, B., Amir, A., Akopyan, F., Andreopoulos, A., Risk, W., Kusnitz, J., Otero, C., Nayak, T., Appuswamy, R., Carlson, P., Cassidy, A., et al., "TrueNorth: Accelerating from zero to 64 million neurons in 10 years," *IEEE Transactions on Computer* **52**, 20–29 (2019).

Table 3. FPGA Resource Utilisation Table.

CLB LUTs	**CLB Registers (234240)**	**CARRY8 (14640)**	**F7 Muxes (58560)**	**F8 Muxes (29280)**	**Block RAM Tile (144)**	**DSPs (1248)**
20517	27097	406	325	52	30.5	253

[5] Nandakumar, S., Kulkarnind, S., Babu, A. V., and Rajendran, B., "Building brain-inspired computing systems: Examining the role of nanoscale devices," *IEEE Nanotechnology Magazine* **12**, 19–35 (2018).

[6] Najem, J., Taylor, G., Weiss, R., Hasan, M., Rose, G., Schuman, C., Belianinov, A., Collier, C., and Sarles, S., "Memristive ion channel-doped biomembranes as synaptic mimics," *ACS Nano* **12:5**, 4702–4711 (2018).

[7] Schuman, C., Kulkarni1, S., Parsa1, M., Mitchell, J., Date, P., and Kay, B., "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science* **2**, 10–19 (2022).

[8] Jo, S., Chang, T., Ebong, I., Bhadviya, B., Mazumder, P., and Lu, W., "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letter* **10**, 1297–1301 (2010).

[9] Kim, M., Park, Y., Kim, I., and Lee, J., "Emerging materials for neuromorphic devices and systems," *iScience* **32**, 1–23 (2020).

[10] Pham, Q. T., Nguyen, T. Q., Hoang, P. C., Dang, Q. H., Nguyen, D. M., and Nguyen, H. H., "A review of SNN implementation on FPGA," in [*2021 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*], 1–6 (2021).

[11] Izhikevich, E., "Simple Model of Spiking Neurons," *IEEE Transactions on Neural Networks* **14**, 1569–1572 (2003).

[12] Shama, F., Haghiri, S., and Imani, M. A., "FPGA realization of Hodgkin-Huxley neuronal model," *IEEE Transactions on Neural Systems and Rehabilitation Engineering* **28**, 1059–1068 (2020).

[13] de Oliveira Neto, J. R., Cajueiro, J. C., and Ranhel, J., "Neural encoding and spike generation for spiking neural networks implemented in FPGA," in [*2015 International Conference on Electronics, Communications and Computers (CONIELECOMP)*], 55–61 (2015).

[14] Guo, W., Fouda, M., Eltawil, A., and Salama, K., "Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems," *Frontiers in Neuroscience* **15**, 1–11 (2021).

[15] Wang, Z., Yu, N., and Liao, Y., "Activeness: A novel neural coding scheme integrating the spike rate and temporal information in the spiking neural network," *Electronics* **12**, 1–11 (2023).

[16] Shouval, H., Wang, S., and Wittenberg, A., "Spike timing dependent plasticity: a consequence of more fundamental learning rules," *Frontiers in Computational Neuroscience* **4:19**, 1–13 (2013).

[17] Gomar, S. and Ahmadi, M., "Digital realization of PSTDP and TSTDP learning," in [*2018 International Joint Conference on Neural Networks (IJCNN)*], 1–5 (2018).

[18] Yousefzadeh, A., Orchard, G., Stromatias, E., Serrano-Gotarredona, T., and Linares-Barranco, B., "Hybrid neural network, an efficient low-power digital hardware implementation of event-based artificial neural network," in [*2018 IEEE International Symposium on Circuits and Systems (ISCAS)*], 1–5 (2018).

[19] Iakymchuk, T., Rosado, A., Frances-Villora, J., and Batallre, M., "Fast spiking neural network architecture for low-cost FPGA devices," in [*7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, Proceedings*], 1–6 (2012).

[20] Huynh, P. K., Varshika, M. L., Paul, A., Isik, M., Balaji, A., and Das, A., "Implementing Spiking Neural Networks on Neuromorphic Architectures: A Review." arXiv, 17 February 2022 `https://arxiv.org/abs/2202.08897`. (Accessed: 31 August 2024).

[21] Isik, M., "A Survey of Spiking Neural Network Accelerator on FPGA." arXiv, 8 July 2023 `https://arxiv.org/abs/2307.03910`. (Accessed: 27 August 2024).

[22] Boutros, A., Yazdanshenas, S., and Betz, V., "You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference," *ACM Transactions on Reconfigurable Technology and Systems* **11**, 1–23 (2018).

[23] Caron, L.-C., Mailhot, F., and Rouat, J., "FPGA implementation of a spiking neural network for pattern matching," in [*2018 IEEE International Symposium on Circuits and Systems (ISCAS)*], 649–652 (2011).

[24] Yasukawa, Z., Okuno, H., Ishii, K., and Yagi, T., "Real-time object tracking based on scale-invariant features employing bio-inspired hardware," *Neural Networks* **81**, 29–38 (2016).

[25] Ju, X., Fang, B., Yan, R., Xu, X., and Tang, H., "An FPGA implementation of deep spiking neural networks for low-power and fast classification," *Neural Computing* **32**, 1–23 (2019).

[26] Diehl, P. U. and Cook, M., "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience* **9**, 99 (2015).