DOI: https://doi.org/10.36910/6775-2524-0560-2026-62-06
UDC 004.31
Kubai Oleh, Postgraduate Student
https://orcid.org/0009-0005-8233-151X
Klyatchenko Yaroslav, PhD in Technical Sciences, Associate Professor
https://orcid.org/0000-0003-4236-4059
National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Kyiv, Ukraine

HARDWARE ACCELERATORS FOR NEURAL NETWORKS: A REVIEW OF THE CURRENT STATE AND PROSPECTS FOR DEVELOPMENT

Kubai O., Klyatchenko Y. Hardware accelerators for neural networks: a review of the current state and prospects for development. This study is aimed to analyze the key aspects of hardware acceleration of neural networks and assess the effectiveness of the modern accelerator frameworks. FPGAs are chosen due to their high reconfigurability and parallelism in comparison to Von Neumann-inspired architectures. The research focuses on hardware-software co-design with the use of programmable logic devices. First, software optimizations are described, including pruning, distillation, quantization, and spiking neural networks as an efficient approach for processing sparse datasets. Common hardware design patterns are discussed, separating optimizations appropriate for dense and sparse inputs, including dataflow patterns and activation passing mechanisms. The research examines the existing FPGA-based accelerators for neural networks comparing their accuracy, hardware complexity, energy efficiency, and ability to perform on-chip training. Transfer learning is identified as a promising research area. Other types of programmable logic devices were analyzed, CPLD is identified as a potential energy-efficient target platform for deploying binarized neural networks. The practical value of the research lies in the potential application of the described principles for deploying neural networks on edge devices.
Keywords: neural networks, neural network acceleration, neuromorphic computing, spiking neural networks, energy efficiency, PLD, FPGA.

Кубай О.Ф, Клятченко Я.М. Апаратні прискорювачі нейронних мереж: огляд поточного стану і перспектив для розвитку. Метою дослідження був аналіз ключових аспектів прискорювачів нейронних мереж на оцінка ефективності сучасних фреймворків для побудови таких прискорювачів. У якості цільової платформи обрано ПЛІС через їх гнучкість та високий паралелізм у порівнянні з чіпами побудованими на основі архітектури фон Неймана. Дослідження фокусуються на спільному проектуванні програмного та апаратного забезпечення. Спочатку розглянуті програмні оптимізації, включаючи квантування, обрізку, дистиляцію та спайкові нейронні мережі як спосіб ефективної обробки розріджених наборів даних. Було описано основні підходи до проектування апаратного забезпечення, включно з шаблонами потоків даних та механізмами передачі активацій нейронів. У роботі розглянуто існуючі прискорювачі нейронних мереж на основі ПЛІС, порівняно точність результатів їх роботи, складність апаратної імплементації, енергоефективність та здатність виконувати алгоритми навчання. Прискорювачі для трансферного навчання були визначені як потенційна сфера для досліджень. Було розглянуто ПЛІС відмінні від FPGA, CPLD було визначено як потенційну енергоефективну платформу для розгортання бінарних нейронних мереж. Практичне значення даного дослідження полягає у потенційному використанні розглянутих принципів для розгортанні нейронних мереж на периферійних пристроях.
Ключові слова: нейронні мережі, прискорення нейронних мереж, нейроморфні обчислення, спайкові нейронні мережі, енергоефективність, ПЛІС.

Introduction. The rapid advancement of artificial intelligence (AI) has led to an unprecedented growth in the computational demands of neural network models. Modern neural networks, particularly deep architectures (deep neural networks, DNN), require substantial processing power and memory bandwidth, which poses significant challenges for conventional general-purpose processors. As a result, hardware accelerators for neural networks have become a crucial research and engineering domain, aiming to improve performance, energy efficiency, and scalability for both training and inference. Among the available hardware platforms – such as CPUs, GPUs, and application-specific integrated circuits (ASICs) – field-programmable gate arrays (FPGAs) occupy a unique and increasingly important position due to their reconfigurability, fine-grained parallelism and energy-performance trade-offs.

FPGAs enable customization of data paths, arithmetic precision, and memory hierarchies to match the computational structure of neural networks. This flexibility is particularly valuable in the context of model compression and algorithm-hardware co-design, which are now central trends in neural network acceleration. Techniques such as network binarization, weight pruning, and knowledge distillation significantly reduce computational complexity, memory footprint, and data movement – factors that often increase energy consumption. When mapped onto FPGA architectures, these methods allow designers to exploit low-bit-width arithmetic, sparse computations and simplified control logic, thereby achieving high throughput and energy efficiency without sacrificing acceptable accuracy.

In parallel, spiking neural networks (SNNs) have emerged as a promising alternative to conventional neural networks, inspired by the event-driven nature of biological neural systems. SNNs naturally align with FPGA architectures due to their temporal sparsity, asynchronous operation, and reliance on simple arithmetic operations such as accumulation ant thresholding. Recent advances in training methods including surrogate gradient learning and ANN-to-SNN conversion, have further increased practical relevance of SNNs for hardware deployment. This article presents a comprehensive review of current hardware accelerators for neural networks with a primary focus on FPGA-based solutions. It analyzes architectural principles, dataflow strategies, and optimization techniques, with particular attention to binarization, pruning, distillation, and spiking neural networks. Furthermore, the article discusses current limitations and outlines the prospects of developments, emphasizing future research directions in energy-efficient, scalable, and adaptive network accelerators built on FPGA platforms.

Research strategy. The research strategy included searching for articles in the Google Scholar database by using the keywords: "hardware accelerators for neural networks", "energy-efficient neural networks", "spiking neural network hardware implementation", "FPGA accelerator for neural network", "neural network and hardware co-design", "programmable logic devices for neural networks" and following references. After, the articles were grouped, and one or several articles were chosen from each group to highlight a specific aspect of neural network acceleration.

Importance of hardware-software co-design. In the previous century humanity created first computers. Machines capable of performing a wide variety of tasks became dominant as they were able to cover the needs of business, which brought investments and allowed them to evolve at a fast rate. Transistors were becoming smaller, reducing the energy consumption and heat dissipation. Gordon Moore noticed this tendency, since then it was known as Moore's law. High-level programming languages appeared which were designed to make programmer's life easier at the cost of less efficient resource utilization, at least in the first compilers. However, it is impossible to shrink transistors forever: first, they cannot be smaller than the size of atoms, second, the lithography process has its limitations. In late 1980s – early 1990s one of these limitations was reached – the wavelength of light. Later ASML overcame this issue by switching to UV light, however today we are facing it again.

Since it is hard to continue scaling transistors down, other approaches for performance improvements are needed. One of the most promising ones is building application-specific accelerators instead of von-Neuman machines. In this approach von-Numan CPU still exists, however it does not execute all programs directly, but rather plays a role of a dispatcher managing the workload across different accelerators. Each accelerator is optimized to execute a specific kind of task.

A popular example of such an accelerator today is a tensor processing unit (TPU). TPUs are systolic arrays optimized for matrix multiplication. Regular CPUs and GPUs can also do it; however, they sacrifice some of the efficiency in the favor of generalization. Being able to execute generic tasks like cryptocurrency mining, and not just AI is one of the reasons why GPUs are still dominant in the AI industry, the companies are still hesitant of the potential of AI. Currently, TPUs can only be found in large quantities in the datacenters built by Google.

Nevertheless, there are an increasing number of papers on application-specific accelerators, including the neural network field, which may become dominant in the future once Von Neumann-inspired architectures cannot keep up with the needs of business.

Software optimizations. AlexNet [1] became a breakthrough in AI in 2012 reaching top-1 and top-5 error rates 37.5% and 17% in the ImageNet LSVRC contest, which was considerably better than the previous state-of-the-art solution. Before 2012, neural networks were not successful in this competition. The difference between this and previous neural networks was in that the authors spent significant time on training the network on GPU, whereas the previous implementations were CPU-based. The improvement was possible because neural networks rely heavily on matrix multiplication which can easily be parallelized.

Another example of how improvements in software can enable further hardware development is the attention mechanism [2] for natural language processing (NLP). Before it, the NLP field was dominated by recurrent neural network (RNN) which were hard to parallelize as each time step is dependent on the previous ones. The attention mechanism enabled the use of transformers and hence better parallelization of training for larger networks.

When talking about hardware-aware software optimizations for neural networks it is important to discern between training and inference. Training requires high precision of weights and outputs, and hence

not much can be done apart from better parallelization and using systolic arrays. When it comes to inference, there are three main optimizations: quantization, pruning and distillation.

Quantization means reducing the precision of inputs, synaptic weights or output activations. Modern GPUs are optimized for quantized networks, for example NVIDIA H100 supports FP8 operations. An extreme case of quantization is binarization which uses 1 bit to store network parameters. Unset bit represents -1, set bit represents +1. A network with binarized inputs, weights and output activations is called fully binarized [3]. Binarization enables the use of XNOR and "popcount" operations which can be effectively implemented with LUT on FPGA instead of summation and multiplication operations [4]. Activation function in such network is replaced with a sign function.

The second optimization is pruning which attempts to minimize the network size by removing unused neurons and connections. This optimization may produce networks with unevenly distributed neurons which makes FPGA a promising target platform for dataflow-style implementations due to their bit-level reconfigurability and high parallelism. For example, in [4] the authors introduce a hardware-aware CNN pruning method for FPGA-based neural accelerators which adaptively prunes a neural network based on the size of a systolic array which is used to compute convolutions.

The third optimization is distillation which aims to train a smaller neural network to behave in a way similar to a larger, original one.

Spiking neural networks. Spiking neural networks (SNN) are frequently called a third generation of artificial neural networks (ANN), however the authors usually discern between ANN and SNN terms. SNN can be thought of as a simplified and hardware-friendly implementation of recurrent neural network. Normally, ANN neuron consists of input weights and activation function, SNN add a neuron membrane potential to this model. Each time inputs are consumed by a neuron, they are added to an internal counter, called a membrane. When the potential reaches a specific threshold, the neuron fires. In literature, the neuron model is frequently discussed in terms of an RC circuit as one of the most efficient implementations.

There are several models of spiking neurons, the most biologically plausible ones are Izhikevich [5] and Hodgkin-Huxley [6], however they are rarely used in practice for more than a proof of concept, as they require a lot of hardware components to implement. More popular are integrate-and-fire (IF) and leaky integrate-and-fire (LIF) models. The models were first described in 1967 [6], however received their names later. IF neuron is the simplest possible case: weights are applied to inputs, then the results are combined and added to a membrane potential, for activation a threshold function is used. LIF extends this model by adding a leakage mechanism – the neuron's potential decays over time. This additional dynamic allows neurons to prioritize more recent information.

In addition to the memory mechanism, SNNs encode information by distributing spikes in time. This leads to improved energy efficiency for sparse inputs and makes SNNs suitable for digital signal processing (DSP). The primary two coding methods are rate-based and latency-based. In rate-based coding the information is encoded with spiking frequency. This increases reliability of outputs, however, negatively affects energy efficiency of neural networks. In latency-based coding the information is encoded in precise spike times. In this case energy efficiency is higher, but the network becomes less reliable. Nowadays researchers are trying to combine both coding schemes [7] to find the middle ground between power consumption and inference accuracy.

Although numerous papers [8–10] show that hardware implementations of SNNs can be more energy-efficient than regular networks, nowadays SNNs are not widespread. There are two reasons for this: they require a specialized ASIC chips or should be implemented on FPGA, and we do not have efficient training algorithms yet. The cause of the second problem lays in a non-differentiable threshold activation function – its derivative is equal to infinity at the origin and to zero for the rest of inputs, this makes it impossible to compute precise gradients for deep networks using backpropagation algorithm. Currently, there are three ways to obtain an SNN model:

1. ANN-to-SNN conversion [11] – ANN is trained using popular frameworks like PyTorch, then converted to SNN. This is the fastest approach yielding the best performance; however, such networks are mostly limited to rate-based coding.
2. Surrogate gradients [12, 13] – a relatively new approach. A non-differentiable threshold function is replaced during a backward pass, this allows to compute approximate gradients for each layer. It was proven empirically that surrogate gradients work, but the theoretical proofs for this training algorithm usually rely on approximation. The best explanation as of today was achieved by

combining a smoothed probabilistic model (SPM) with stochastic automatic differentiation (StochAD) [14].

3. Biologically inspired algorithms – usually rely on neuroplasticity rules implemented in hardware. Due to their simplicity and low memory requirements, these algorithms can be trained on specialized hardware like Intel Loihi [15], whereas other algorithms usually require off-chip learning. The most popular algorithm is Spike-Timing-Dependent Plasticity (STDP) in which the strength of connections between neurons is controlled by their relative firing order.

Overall, spiking neural networks have better energy efficiency than ANNs, can be implemented using hardware components like RC circuits or counters, and work well for DSP. However, there are several issues that prevent SNNs from replacing binarized neural networks (BNN): slow training algorithms and the absence of cost-efficient hardware. FPGAs are normally used to run such networks; however, the flexibility comes at an additional cost in comparison to specialized chips.

Hardware architectural patterns. Most of papers on neural network acceleration use FPGA for either prototyping or even implementation. Both applications are motivated by high reconfigurability, parallelism and cost lower than building an ASIC. It is also worth mentioning existing ASICs [15–17] which are faster and more energy-efficient than FPGAs, however, less flexible and either do not support learning algorithms at all or support the most basic ones. Regardless of the platform, there are several key architectural choices that affect the performance of an accelerator for a specific type of neural networks: activation passing mechanism and data flow patterns.

The first aspect to discuss is activation passing, or how data moves through an accelerator. The chips operate in either synchronous or asynchronous mode. Asynchronous mode does not necessarily mean that the underlying hardware does not have a global clock, but rather that the neural signals are not batched together and are routed at the occurrence time, asynchronously. This means that each neural signal should either be sent via a specialized path, either physical or virtual. Most SNN accelerators with asynchronous flow use virtual paths by using packets, as physical paths are more expensive to implement in hardware. Each packet contains additional information on top of a signal itself, which may be inefficient in terms of storage for dense datasets in comparison to arrays, however it is a good for sparse data. The most common packet format is Address-Event Representation (AER), used in Loihi [15], IBM NorthPole [17] and SpiNNaker [18] accelerators among others. AER packet contains the following fields: address of a neuron that fired, timestamp of the firing event, type. A packet may be extended with other properties depending on the needs of a specific accelerator.

A different approach is passing inputs as a matrix. In this case no additional addressing information is needed since neuron addresses are encoded by indexes of elements in the matrix. However, this approach introduces a potentially large compute and storage overhead when dealing with highly sparse datasets, for example, outputs of event cameras. The approach is popular in modern ANNs as they normally process dense data.

A second important aspect is data flow patterns: what data flows through an accelerator and what remains stationary in its processing elements (PE). The most common patterns are weight-stationary, input-stationary, output-stationary, and row-stationary. These patterns allow to minimize data movement for specific neural network architectures, however, may be inefficient for some other models. Picking an appropriate pattern may allow to execute a neural network on a chip with limited global or PE-local memory.

In the weight-stationary pattern synaptic weights are loaded into local registers of processing elements, inputs are streamed through the accelerator. This approach works well for deep neural networks with relatively low input and output sizes in comparison to the number of weights if all the weights can fit into memory, that is why it is commonly used in TPUs built by Google [19].

In the input-stationary pattern input activations are stored in PE registers, weights and inputs are streamed through an accelerator. This pattern works well for CNNs layers with a lot of filters, meaning that each input value is used frequently. The drawbacks of this dataflow pattern are the fact that it may require a high bandwidth for moving weights and an additional computational tree to obtain outputs.

The third pattern is output-stationary, where partial sums are stored in registers of processing elements, weights and inputs are streamed through. This minimizes the movement of partial sums, which often have high precision even in quantized networks, including expensive write operations to global memory.

The final and the most complex pattern is row-stationary [20] where a row of 1D convolution is assigned to a processing element. Weights and partial sums remain stationary, inputs are streamed through an accelerator. This approach is considered the most energy-efficient for convolutional neural networks as it maximized the local reuse of all three data types: weights, inputs and outputs.

In the next sections we discuss existing FPGA-based accelerators, other types of programmable logic arrays, and ASICs. Then we compare them.

Application-specific integrated circuits (ASIC). Currently, the AI field is dominated by GPUs due to investor's hesitation regarding the AI's potential and the fact that GPUs can be used for purposes other than AI in case the field stops growing. Despite the fact, some ASICs accelerators for AI already exist, some of them will be discussed in this chapter.

Loihi [15] is a neuromorphic chip designed by Intel to mimic brain-like processing. The chip is composed of 128 neuromorphic cores, 3 x86 cores and 33Mb of SRAM fabricated using 14nm technology. Each of the 128 neuromorphic cores consists of 1024 simple spiking neural units grouped into trees. The chip also contains external interfaces which can be used to either read inputs, produce outputs or connect the accelerators into a larger mesh. This accelerator is specifically designed for SNNs and supports only LIF neuron model. Apart from inference, it can execute simple STDP-like learning algorithms.

The first generation of Loihi chip was limited in terms of neuron logic and hence has a little importance to researchers, therefore a new architecture, Loihi 2, was designed and implemented. Similarly to the previous generation, it is composed of 128 neuromorphic cores and extension interfaces, but 6 x86 cores. Unlike the previous version, it supports user-defined arithmetic and logic allowing to implement neuron models other than LIF.

TrueNorth [21] is an SNN accelerator designed by IBM. The chip consists of 4096 cores allowing for up to 1 million neurons and 256 million synapses. The chip has shown 50-100 times faster processing on sparse datasets in comparison to GPUs, including video object detection and other tasks, and its power consumption can be reduced to 65mW at 0.75V input, but unlike Loihi, it only supports inference.

NorthPole [17] is a newest chip from IBM which heavily relies on memory intertwined with compute logic. This reduces the overhead of data movements, however, requires more meticulous design of neural networks. The accelerator is implemented using 12nm process, it contains 256 cores, containing their own memory and compute units. Each core can perform 2048 8-bit operations per second. The chip supports 8-bit, 4-bit and 2-bit integer operations as well as 16-bit floating-point operations, allowing for denser execution of quantized networks. Similarly to TrueNorth, the chip only supports inference.

SpiNNaker 2 [22] is a chip developed by the Dresden University of Technology. It contains 153 ARM cores with 19mb of on-chip SRAM, 2gb of RAM, and AI accelerators manufactured with 22nm process. Unlike SpiNNaker 1, which was built exclusively for SNN, in SpiNNaker 2 each ARM core is coupled with a Multiply-Accumulate (MAC) unit, and hence it supports classical and event-driven ANNs. Given that both chips were designed for flexibility in simulating brain-like activities in real-time, it is expected that the chips perform worse than GPUs or Vitis AI-powered FPGAs for ANNs, however excel in executing SNNs.

A side-by-side comparison of the chips is provided in the Table 1. Max neuron and max synapses counts are not specified for the IBM NorthPole chip as they may vary based on how exactly a neural network is mapped onto the chip.

Table 1. Comparison of ASICs for neural network acceleration

Name	**Max neurons**	**Max synapses**	**Supports on-chip learning**	**Average power**	**Release year**
Intel Loihi	128k	128m	Yes	<1.5W	2018
Intel Loihi 2	1m	120m	Yes	<1W	2021
IBM TrueNorth	1m	256m	No	<0.3W	2014
IBM NorthPole	256 cores	224mb SRAM	No	~74W	2023
SpiNNaker 2	152k	152m	Yes	2-5W	2021

Choosing an FPGA for research. Before describing specific architectures of neural network accelerators, it is important to describe how to choose an appropriate device to perform testing on. Based on the requirements of a specific research, the important aspects are:

1. LUT, DSP count – binarized neural networks may be implemented using only LUT, however high-precision activations may benefit from DSP slices.
2. Support for Vitis AI – this proprietary framework is frequently used as a performance baseline when presenting a new architecture, however it requires a high-end board. The most accessible board is Kria KV260 Vision AI Starter Kit [23], it is built for developing vision applications for edge devices.
3. Ability to execute python scripts – AUP PYNQ-Z2 [24] can host a Jupyter Notebook web server, pre-compiled hardware overlays can be loaded similarly to standard python libraries.
4. Existing libraries and community - AUP PYNQ-Z2 [24] is a popular university board with a lot of publicly available projects, making it easier to reuse the existing work. FINN documentation [25] describes how to configure the board to run the framework, making it easier for researchers to run inference on the existing BNN framework for comparison.
5. Ability to accurately measure power of FPGA chip – development boards usually consist of several components like ARM processors, FPGA chip, memory, sensor controllers. While measuring power consumption of a specific architecture these components may affect the results, so, if possible, power consumption should be measured for only FPGA chip. This, however, requires separate power rails for the FPGA chip, and, optionally, power controllers. An example of such board is AMD Zynq™ UltraScale+™ MPSoC ZCU104 Evaluation Kit [26].

FPGA-based implementations. The first FPGA-based framework for neural networks worth mentioning is Vitis AI. It was developed by Xilinx, FPGA manufacturer acquired by AMD in 2023 to accelerate ANNs. The core of the framework is a Deep-Learning Processor Unit (DPU), it is not a physical element in FPGA, but rather an architecture that can be implemented using LUT, DSP, FF, and BRAM blocks. A common workflow usually consists of two stages: development and deployment, with testing between individual steps. The framework contains the following software to aid the model development:

1. Vitis AI Model Zoo includes pretrained models optimized for Xilinx FPGAs.
2. Vitis AI Model Inspector is used to perform initial sanity checks on a model.
3. Vitis AI Optimizer adapts models for sparse datasets.
4. Vitis AI Quantizer converts weights and activations from 32-bit floating point to 8-bit integer format.
5. Vitis AI Compiler performs optimizations and maps a quantized neural network onto an FPGA in a dataflow style.

When model is implemented and tested, the following software can be used to ease the deployment process:

1. Vitis AI Runtime managers AI job execution on DPU, integrates with C++ or Python software.
2. Vitis AI Library contains a set of high-level APIs built on top of Vitis AI Runtime to simplify model deployment.
3. Vitis AI Profiler provides real-time monitoring for deployed models.

The framework supports the following FPGA boards: AMD Zynq™ UltraScale+™ MPSoCs, the Kria™ KV260 [23], Versal™ and Alveo. It mostly uses weight-stationary dataflow with elements of output-stationary to efficiently combine layer outputs. [27] compares the framework to CPU and GPU-based neural network inference, achieving 3.33-5.82x throughput and 3.39-6.30x energy efficiency improvement on FPGA in comparison to CPU, 2.62-4.58x throughput and 14.12-26.26 energy efficiency improvement when comparing FPGA to Nvidia GPU. Vitis AI is built for regular neural networks and has lower power efficiency than SNN-based frameworks as shown in Table 2.

Another framework for building ANN accelerators is FINN [28] which is optimized for fully binarized CNN inference. Vivado High-Level Synthesis (HLS) was used for the implementation including the following optimizations, which are possible due to full binarization, and are combined in the main building blocks of this framework – Matrix-Vector Threshold Units (MVTU):

- "Popcount" operation is used instead of an adder tree for combining neuron inputs.
- Batch normalization for outputs.
- Boolean OR is used for MaxPool layers instead of complex comparators.
- XNOR to combine weights with inputs.

Table 2 shows that FINN achieves better power efficiency and significantly higher throughput than the AMD proprietary Vitis AI framework. This can be explained by the fact that the FINN framework adopts fully binarized networks, whereas Vitis AI primarily relies on 8-bit quantization which requires more

expensive hardware components. Next in this section SNN frameworks are described which can bring power consumption event lower for sparse datasets, however increasing latency due to the coding mechanisms. FPGAs normally have high static power consumption, so, taking into account that different papers used different chips for measuring power, the collected data may not be reliable, however, it cannot be improved by subtracting static power as it may vary depending on actual conditions during an experiment. Nevertheless, the data from Table 2 for SNN-specific ASICs, Intel Loihi and IBM TrueNorth, is a good approximation of the improvement in power consumption that can be achieved by switching to SNN models. One important note regarding this data is that the accelerators were tested on datasets like MNIST and CIFAR-10 which have relatively low sparsity in comparison to what data captured by a deployed event-based camera may look like, so theoretical power consumption may be even lower. The original paper [28] implements the architecture on AMD Zynq™ 7000 SoC ZC706 Evaluation Kit, which has a limited AI software support, but later a comparison between Vitis AI and FINN [29] frameworks was done on Kria KV260 Vision AI Starter Kit [23] showing 2 to 15 times improvement of energy efficiency, 1.06 to 4.67 times higher throughput, and 1.2 to 10.47 times lower latency for FINN framework.

SIES (Spiking Neural Network Inference Engine for SCNN) [30] is an FPGA-based inference hardware accelerator for spiking convolutional neural networks (SCNN). The authors used an ANN-to-SNN conversion algorithm to obtain an SCNN with rate coding and possibly lower power efficiency than other frameworks, however the information on energy efficiency was not reported. The architecture is based on a systolic array to compute membrane potential and includes hardware-based modules for max-pooling and convolutional layers. It was implemented on Xilinx Virtex Ultrascale (VU440) XCVU440 FPGA, supporting up to 4000 neurons and 256000 synapses.

Another SNN inference framework relying on rate coding and ANN-to-SNN conversion is Minitaur [31], which was implemented on Xilinx Spartan-6 LX150 FPGA. Unlike some other implementations mentioned in this paper, Minitaur is not based on a systolic array and allows executing neural networks of different shapes. The accelerator was tested using a 4-layer fully connected with 784 input neurons, 2 hidden layers with 500 neurons and 128 output neurons on MNIST dataset achieving 92% accuracy. It achieves relatively low power consumption in comparison to other accelerators even with rate coding, however this can be explained by the usage of a low-cost Spartan 6 FPGA and small custom neural networks, whereas other papers are using production grade DNNs.

As discussed previously, SNNs with rate coding produce more stable results, however, require more power and have higher latency. Cerebron [32] is built for rate-coded SNNs using surrogate gradient for train neural networks, and hence achieving the best power efficiency among the frameworks compared in Table 2. It is a weight-stationary SNN inference accelerator designed to fully exploit spatiotemporal sparsity, using a reconfigurable event-driven compute engine and AER event format. The architecture is composed of a controller, buffers, address generator and a compute engine. The buffer is used to cache membrane potential and weights. The address generator produces weight and membrane potential addresses. The compute engine is composed of processing elements used to execute logic of individual neurons.

E3NE [33] is an end-to-end framework for building FPGA-based rate-coded SNN accelerators for edge devices by compiling PyTorch models into a bitstream. The framework optimizes the usage of logic blocks by maximizing parallelism. The paper concentrates on support for "emerging neural encoding schemes", meaning approaches other than rate-based coding, for example latency-based. According to the authors, they were able to achieve 50% decrease in required hardware resources and 20% better power efficiency.

FireFly [34] is an FPGA-based accelerator for spiking convolutional neural networks (SCNNs) using weight-stationary dataflow. This is the first accelerator that introduces DSP block optimizations. The authors analyzed the neural dynamics of spiking neurons and used multiplex-accumulate operation implemented with DSP48E2 block on a Xilinx Ultrascale FPGA. The accelerator achieves higher performance than other architectures which rely solely on LUT.

Most FPGA-based SNN accelerators assume high sparsity in spikes, in this case a network can be implemented using a systolic array improving the energy efficiency. However, the authors of S2N2 framework [35] showed that this assumption does not hold true for applications with signals of large temporal dimension, for example radio frequency (RF) signal processing. Optimizations introduced by the framework:

- Replaced Address Event Representation (AER) packets with binary tensors. AER packets contain neuron addresses which may be not memory-efficient when the spike sparsity is low.
- Tick-batching, which is normally used to process AER events more efficiently, is eliminated in S2N2.
- Support for axonal and synaptic delays between the network layers
- RF samples are turned into sparse events in the In-Phase/Quadrature (I/Q) plane.

Another SNN accelerator which combines spikes across timestamps is SpinalFlow [36]. The accelerator compresses sorted spike sequences and uses 4-bit quantization for inputs. Neurons are emulated by an array of processing elements based on Eyeriss [20] architecture. Each processing element is reconfigurable and can execute in one of several modes, including convolution and max pooling.

And the final SNN framework for dense datasets is SyncNN [37] which attempts to increase throughput by batching events and processing them events layer-by-layer, achieving better performance but introducing delays in processing, which may be an issue for leveraging the approach in real-time systems. Neuron spikes on a final step several times. Uses mixed-precision quantization (first few layers 8-bit, middle – 4-bit, last few layers – 8-bit).

Binarized neural networks are fast and energy-efficient for inference, however they perform worse than their full-precision alternatives when many layers are required. To resolve this issue a piecewise linear approximation was suggested: a function inputs are split in ranges, each range is approximated using a linear function. Approaches that can be used for selecting the range bounds and the function coefficients are discussed in several papers, among them [38] proposes a shift-based implementation which replaces weight multiplications and activation functions with arithmetic shifts. Such approach limits the possible weight values, however, allows more performant and energy-efficient implementation of neural networks.

[7] investigates the use of spiking neural networks or edge devices. The paper uses a first-order LIF neurons to implement a vision-based model on Xilinx Artix-7 FPGA. The network is implemented in dataflow style: inputs are combined with a cascaded adder network and then passed to a LIF neuron hardware unit which is implemented using a shift register. The model is then compared to the state-of-the-art binarized convolutional neural network (BCNN), SNN shows 86% higher energy efficiency.

Table 2. Accelerator performance, "?" – unknown

Accelerator	**Device**	**Frequency**	**Dataset**	**Model**	**Accuracy (%)**	**Latency (µs)**	**Throughput (fps)**	**Power (W)**
Cerebron	XC7Z100	200	MNIST	ConvNet	99.40	26	38,500	1.4
			CIFAR-10	MobileNet	91.90	10630	90	1.4
			MLND-Capstone	SegNet	97.30	800	1,250	1.4
E3NE	XCVU13P	200	MNIST	CNN[1]	99.3	409	2,445	3.6
			MNIST	LeNet-5	99.1	294	3,400	3.4
		150	CIFAR-10	AlexNet	80.6	70,000	14,3	4.7
			CIFAR-100	VGG-11	65.0	163,000	6.1	5.7
FINN	ZC706	200	MNIST	SFC	95.83	0.31	12,361,000	7.3
			MNIST	LFC	98.4	2.44	1,561,000	8.3
			CIFAR-10	CNV	80.1	283	21906	3.6
			SVHN	CNV	94.9	283	21906	3.6
FireFly	XCZU3EG	300	MNIST	SCNN-5	98.12	?	2036	2.55
			CIFAR-10	SCNN-7	91.36	?	966	2.55
			CIFAR-100	SCNN-11	64.28	?	470	2.55
			DVS-CIFAR10	SCNN-9	72.4	?	282	2.55
			DVS-Gesture	SCNN-9	89.29	?	282	2.55

Minitaur	Spartan-6	?	MNIST	FC[2]	92	?	?	1.5
			Newsgroup	?	71	?	?	1.5
SIES	XCVU44	200	CIFAR-10	VGG-16	91.46	?	?	?
S2N2	ZYNQ	?	RadioML. 2016	CNN[3]	91	?	?	?
	ZCU111	?	RadioML. 2018	CNN[4]	68.5	?	?	?
Vitis AI	ZCU-104	300	CIFAR-10	?	58.76	?	1021.45	60
TrueNorth	TrueNorth	No global clock	MNIST	CNN	99.4	1,000	1000	0.2
Loihi	Loihi	0.01	MNIST	CNN[5]	94.7	10,000	97	0.24

[1] 28×28-32C3-P2-32C3-P2-256-10
[2] 784-500-500-10
[3] 16x16-16C5-8C5-128LIF-11LIF-11Softmax
[4] 16×16-64C5-64C5-128C3-128C3-1024-24Softmax
[5] 32×32-16C5-P2-8C3-P2-100-10

Table 3 summarizes the applications of the neural network accelerators. Most of them support only inference with only Intel Loihi supporting simple learning rules which may work for transfer learning, for example. However, the chip is not publicly available.

Table 3. Accelerator applications

Application	**Accelerators**
Inference	Cerebron, E3NE, Eyeriss, FINN, FireFly, Loihi, Minitaur, SIES, SyncNN, SpinalFlow, S2N2, TrueNorth, Vitis AI (All)
Local training rules	Loihi
Backpropagation-based training	(None among mentioned)

According to the FPGA resource usage information from Table 4 and the accelerator performance information from the Table 3, non-binarized neural networks benefit from leveraging DSP blocks. On the other hand, accelerators for fully binarized neural networks, for example FINN, achieve better performance using only LUT, however sacrificing some of the inference accuracy.

Table 4. FPGA resource usage, "?" – unknown, "-"– not applicable, "~"– estimation

Accelerator	**Model**		**LUT**	**FF**	**DSP**	**BRAM (blocks)**
	Neurons	**Parameters**				
Cerebron	-	-	85,926	70,544	0	283
E3NE	~32,762	~217,194	44,000	36,000	?	?
	~9,118	~60,866	27,000	24,000	?	?
	~935,323	~60,965,224	48,000	50,000	?	?
	~9,116,136	~132,863,336	88,000	84,000	?	?
FINN	~1,306	~269,322	91131	?	?	4.5
	~2,842	~1,863,690	82988	?	?	396
	~32,778	~3,510,922	46253	?	?	186
FireFly	-	-	15,000	?	288	162
Minitaur	~1,794	~648,010	22,000	?	?	?
S2N2	~3,211	~70,707	~27,664	~11,704	~11	~41
	~21,272	~2,448,344	~65,780	~34,000	<42	~1058
Vitis AI	-	-	105,845	198,725	1,420	210

Other programmable logic devices. Most accelerators are built using FPGAs as they allow to implement and reconfigure architectures with least effort. However, FPGAs have some issues, the most prominent among them is relatively high current leakage in volatile block RAM (BRAM) which leads to high static power consumption. However, this can be alleviated by using NV-FPGA based on non-volatile BRAM [39]. In this section we discuss other types of programmable logic devices and compare them to FPGA. The first generation of such chips is Simple Programmable Logic Devices (SPLD) which are built from simple logic gates like OR and AND, programmable connections between the gates, and flip-flops. Such devices are small, cost-effective, and energy-efficient, however, hard or impossible to reprogram, which makes them inapplicable for most production-ready models. They lack DSP blocks to accelerate multiply-and-accumulate (MAC) operations, however, can still be run binarized neural networks efficiently. Most common types of such devices are:

1. Programmable Read-Only Memory (PROM) is a digital memory device which contains a fixed AND array and an OR array that can be programmed only once after manufacturing by burning fuses. There are also other types of PROM which can be reprogrammed multiple times: Erasable PROM (EPROM) which can be erased with UV light, and Electrically Erasable PROM (EEPROM). There is little research on deploying neural networks on such devices, mostly concentrating on EEPROM as it is the easiest to work with, however some industries may benefit from using EPROM as it is more resistant to radiation than regular CPUs or FPGAs.
2. Programmable Logic Arrays (PLA) is the most complex type of device which allows reconfiguring both, OR and AND arrays.
3. Programmable Array Logic (PAL) has a reconfigurable AND array and a fixed OR array. It was developed to overcome certain disadvantaged of PLA such as longer delays caused by two reconfigurable arrays. PLA and PAL are no longer used in neural network accelerator research because of their small, fixed number of input pins and the absence of DSP blocks.
4. Generic Array Logic (GAL) is like PAL in that it contains a fixed OR array and a programmable AND array, however it also has programmable outputs and can be reprogrammed. GAL are usually too small to host a usable neural network, and they were developed only a few years before FPGAs, so AI researchers normally do not use this type of device.

The second generation of PLD is Complex Programmable Logic Devices (CPLD) are chips with higher complexity than SPLD. This type of PLD was developed later than FPGA and given FPGA's better capabilities for working with floating-point numbers, CPLDs didn't become popular in the area and were used only in a small number of papers for very simple neural networks [40]. However, they might still be interesting for building AI accelerators as they usually store configuration in non-volatile memory. Moreover, after binarized neural networks were proposed in 2016 [3], accelerators no longer need to support floating point or integer arithmetic as such networks can be implemented purely based on Boolean operations.

The final PLD category discussed in this section is Coarse-Grained Reconfigurable Arrays (CGRA). While FPGAs are reconfigurable on a bit level, CGRAs are reconfigurable on a word level which makes them more energy-efficient and potentially faster than FPGAs. CGRAs are composed of an array of interconnected processing elements, each one operating on word-level data. Some FPGA-based accelerators like Cerebron [32] are architecturally similar to CGRAs, which suggests that a LUT-based design of FPGAs might be an overkill in some cases. CGRA is the second most common PLD after FPGA, researched in modern papers on neural network acceleration. However, unlike FPGAs which are mostly used for prototyping, CGRAs are considered a target platform for production deployments.

Conclusions. In this paper we discussed existing accelerator frameworks for neural networks, in particular – SNN for handling sparse data. Most researchers prefer FPGAs for either prototyping or final implementation due to their reconfigurability and lower prices in comparison to ASICs. Several areas for improvements were identified, the most notable one is the fact that most of these accelerators are for inference and either do not support training or support only simplest, Hebbian-like algorithms, for example STDP. The models are usually trained using popular frameworks like PyTorch or TensorFlow and then implemented in hardware. Training is hard to optimize because the algorithms normally require full-precision arithmetic and thus the networks cannot be quantized. However, end-to-end training becomes less popular these days as there are a lot of existing models which can be reused via a process called transfer learning, however cost and energy-efficient accelerators for this area are not in the focus of researchers yet.

Second possible area of improvement is inference optimization. FPGAs allow for faster inference by implementing networks in dataflow style, eliminating the shortcomings of the fetch-decode-execute cycle. In comparison to GPU and TPU-based implementations this approach has another advantage: redundant neurons can be eliminated or merged; the network is not required to have perfectly rectangular matrices. Most of the papers mentioned above introduce common frameworks for a specific class of neural networks, but the accelerator characteristics can be improved even further by considering a specific network model. The future research areas may include:

- Optimize the network layout on a chip so the processing elements emulating the neurons which communicate more frequently are located closer to each other.
- Processing elements reuse algorithms to minimize data movement.
- Inference on larger networks – most of the works are limited to MNIST, CIFAR-10 and CIFAR-100 datasets, only some have discussed deeper networks.
- Accelerate BNN inference on CPLD.
- Theoretical frameworks for platform-independent comparison of accelerator energy consumption in the context of SNNs.

And the final possible area is a theoretical justification for SNN training algorithms which may be beneficial for future works in the hardware-software co-design field.

References:

1. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks. Communications of the ACM. 2017. Vol. 60, No. 6. P. 84–90. DOI: https://doi.org/10.1145/3065386.
2. Vaswani A., Shazeer N., Parmar N. et al. Attention is all you need. Advances in neural information processing systems. 2017. DOI: https://doi.org/10.48550/arXiv.1706.03762.
3. Courbariaux M., Hubara I., Soudry D. et al. Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1 [Electronic resource]. arXiv, 2016. Available at: https://arxiv.org/abs/1602.02830. DOI: https://doi.org/10.48550/arXiv.1602.02830.
4. Zhao R., Song W., Zhang W. et al. Accelerating binarized convolutional neural networks with software-programmable FPGAs. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17). ACM, 2017. P. 15–24 DOI: https://doi.org/10.1145/3020078.3021741.
5. Izhikevich E. M. Simple model of spiking neurons. IEEE Transactions on Neural Networks. 2003. Vol. 14, No. 6. P. 1569–1572. DOI: https://doi.org/10.1109/TNN.2003.820440.
6. Hodgkin A. L., Huxley A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of Physiology. 1952. Vol. 117, No. 4. P. 500–544. DOI: https://doi.org/10.1113/jphysiol.1952.sp004764.
7. Wang Z., Yu N., Liao Y. Activeness: a novel neural coding scheme integrating the spike rate and temporal information in the spiking neural network. Electronics. 2023. Vol. 12, No. 19. P. 3992. DOI: https://doi.org/10.3390/electronics12193992.
8. Ali A. H., Navardi M., Mohsenin T. Energy-aware FPGA implementation of spiking neural network with LIF neurons [Electronic resource]. arXiv, 2024. Available at: https://arxiv.org/abs/2411.01628. DOI: https://doi.org/10.48550/arXiv.2411.01628.
9. Yan Z., Bai Z., Wong W.-F. Reconsidering the energy efficiency of spiking neural networks [Electronic resource]. arXiv, 2025. Available at: https://arxiv.org/abs/2409.08290. DOI: https://doi.org/10.48550/arXiv.2409.08290.
10. Li T., Li J., Shen G. et al. FireFly-T: high-throughput sparsity exploitation for spiking transformer acceleration with dual-engine overlay architecture [Electronic resource]. arXiv, 2025. Available at: https://arxiv.org/abs/2505.12771. DOI: https://doi.org/10.48550/arXiv.2505.12771.
11. Bu T., Fang W., Ding J. et al. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks [Electronic resource]. arXiv, 2023. Available at: https://arxiv.org/abs/2303.04347. DOI: https://doi.org/10.48550/arXiv.2303.04347.
12. Neftci E. O., Mostafa H., Zenke F. Surrogate gradient learning in spiking neural networks [Electronic resource]. arXiv, 2019. Available at: https://arxiv.org/abs/1901.09948. DOI: https://doi.org/10.48550/arXiv.1901.09948.
13. Eshraghian J. K., Ward M., Neftci E. et al. Training spiking neural networks using lessons from deep learning [Electronic resource]. arXiv, 2023. Available at: https://arxiv.org/abs/2109.12894. DOI: https://doi.org/10.48550/arXiv.2109.12894.
14. Gygax J., Zenke F. Elucidating the theoretical underpinnings of surrogate gradient learning in spiking neural networks [Electronic resource]. arXiv, 2024. Available at: https://arxiv.org/abs/2404.14964. DOI: https://doi.org/10.48550/arXiv.2404.14964.
15. Davies M., Srinivasa N., Lin T.-H. et al. Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro. 2018. Vol. 38, no 1. P. 82–99. DOI: https://doi.org/10.1109/MM.2018.112130359.
16. Akopyan F., Sawada J., Cassidy A. et al. TrueNorth: design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2015. Vol. 34, No. 10. P. 1537–1557. DOI: https://doi.org/10.1109/TCAD.2015.2474396.
17. Cassidy A. S., Arthur J. V., Akopyan F. et al. 11.4 IBM NorthPole: an architecture for neural network inference with a 12nm chip. Proceedings of the 2024 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2024. P. 214–215. DOI: https://doi.org/10.1109/ISSCC49657.2024.10454451.

18. Furber S. B., Galluppi F., Temple S. et al. The SpiNNaker project. Proceedings of the IEEE. 2014. Vol. 102, no 5. P. 652–665. DOI: https://doi.org/10.1109/JPROC.2014.2304638.
19. Jouppi N. P., Young C., Patil N. et al. In-datacenter performance analysis of a tensor processing unit [Electronic resource]. arXiv, 2017. Available at: https://arxiv.org/abs/1704.04760. DOI: https://doi.org/10.48550/arXiv.1704.04760.
20. Chen Y.-H., Krishna T., Emer J. et al. 14.5 Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2016. P. 262–263. DOI: https://doi.org/10.1109/ISSCC.2016.7418007.
21. Merolla P. A., Arthur J. V., Alvarez-Icaza R. et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. Science. 2014. Vol. 345, No. 6197. P. 668–673. DOI: https://doi.org/10.1126/science.1254642.
22. Mayr C., Hoeppner S., Furber S. SpiNNaker 2: a 10 million core processor system for brain simulation and machine learning [Electronic resource]. arXiv, 2019. Available at: https://arxiv.org/abs/1911.02385. DOI: https://doi.org/10.48550/arXiv.1911.02385.
23. Kria KV260 Vision AI Starter Kit [Electronic resource] / AMD. Available at: https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html (accessed: 24.02.2026).
24. AUP PYNQ-Z2 [Electronic resource]. AMD. Available at: https://www.amd.com/en/corporate/university-program/aup-boards/pynq-z2.html (accessed: 01.02.2026).
25. FINN framework documentation [Electronic resource] / AMD. Available at: https://finn.readthedocs.io/en/latest/getting_started.html (accessed: 24.02.2026).
26. AMD Zynq™ UltraScale+™ MPSoC ZCU104 Evaluation Kit [Electronic resource] / AMD. Available at: https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/zcu104.html (accessed: 01.02.2026).
27. Li Z., Hong F. Z., Yue C. P. FPGA-based acceleration of neural network for image classification using Vitis AI [Electronic resource]. arXiv, 2024. Available at: https://arxiv.org/abs/2412.20974. DOI: https://doi.org/10.48550/arXiv.2412.20974.
28. Umuroglu Y., Fraser N. J., Gambardella G. et al. FINN: a framework for fast, scalable binarized neural network inference. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17. ACM, 2017. P. 65–74. DOI: https://doi.org/10.1145/3020078.3021744.
29. Urbano Pintos N., Lacomi H., Lavorato M. Comparación de Vitis-AI y FINN para implementar redes neuronales convolucionales en FPGA. Elektron. 2024. Vol. 8, No. 2. P. 61–70. DOI: https://doi.org/10.37537/rev.elektron.8.2.200.2024.
30. Wang S.-Q., Wang L., Deng Y. et al. SIES: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array. Journal of Computer Science and Technology. Vol. 35, No. 2. P. 475–489. DOI: https://doi.org/10.1007/s11390-020-9686-z.
31. Neil D., Liu S.-C. Minitaur, an event-driven FPGA-based spiking network accelerator. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2020. Vol. 22, no, 12. P. 2621–2628. DOI: https://doi.org/10.1109/TVLSI.2013.2294916.
32. Chen Q., Gao C., Fu Y. Cerebron: a reconfigurable architecture for spatiotemporal sparse spiking neural networks. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2022. Vol. 30, No. 10. P. 1425–1437. DOI: https://doi.org/10.1109/TVLSI.2022.3196839.
33. Gerlinghoff D., Wang Z., Gu X. et al. E3NE: An end-to-end framework for accelerating spiking neural networks with emerging neural encoding on FPGAs. IEEE Transactions on Parallel and Distributed Systems. 2021. P. 1. DOI: https://doi.org/10.1109/TPDS.2021.3128945.
34. Li J., Shen G., Zhao D. et al. FireFly: A high-throughput hardware accelerator for spiking neural networks with efficient DSP and memory optimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2023. Vol. 31, No. 8. P. 1178–1191. DOI: https://doi.org/10.1109/TVLSI.2023.3279349.
35. Khodamoradi A., Denolf K., Kastner R. S2N2: a FPGA accelerator for streaming spiking neural networks. Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'21). ACM, 2021. P. 194–205. DOI: https://doi.org/10.1145/3431920.3439283.
36. Narayanan S., Taht K., Balasubramonian R. et al. SpinalFlow: an architecture and dataflow tailored for spiking neural networks. Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA) IEEE. 2020. P. 349–362. DOI: https://doi.org/10.1109/ISCA45697.2020.00038.
37. Panchapakesan S., Fang Z., Li J. SyncNN: evaluating and accelerating spiking neural networks on FPGAs. ACM Transactions on Reconfigurable Technology and Systems. 2022. Vol. 15, No. 4. P. 1–27. DOI: https://doi.org/10.1145/3514253.
38. Zhang G., Li B., Wu J. et al. A low-cost and high-speed hardware implementation of spiking neural network. Neurocomputing. 2020. Vol. 382, 03.2020. P. 106–115. DOI: https://doi.org/10.1016/j.neucom.2019.11.045.
39. Zhang H., Zhao M., Zheng H. et al. Towards high-throughput neural network inference with computational BRAM on nonvolatile FPGAs. Proceedings of the 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2024. P. 1–6. DOI: https://doi.org/10.23919/DATE58400.2024.10546738.
40. Hern M., Leal R., Aguilera-Galicia C. An artificial neural network on a complex programmable logic device as a virtual sensor. Proceedings of Second International Workshop on design of Mixed-Mode Integrated and Applications. 1998.

Історія статті:
Отримано: 10.02.2026 Доопрацьовано: 25.02.2026 Прийнято до друку: 23.03.2026 Опубліковано: 29.03.2026