A Survey of Hardware Self-Organizing Maps

Slaviša Jovanović, *Member, IEEE*, and Hiroomi Hikawa, *Member, IEEE*

***Abstract*— Self-organizing feature maps (SOMs) are commonly used technique for clustering and data dimensionality reduction in many application fields. Indeed, their inherent property of topology preservation and unsupervised learning of processed data without any prior knowledge put them in the front of candidates for data reduction in the Internet of Things (IoT) and big data (BD) technologies. However, the high computational cost of SOMs limits their use to offline approaches and makes the online real-time high-performance SOM processing more challenging and mostly reserved to specific hardware implementations. In this article, we present a survey of hardware (HW) SOM implementations found in the literature so far: the most widely used computing blocks, architectures, design choices, adaptation, and optimization techniques that have been reported in the field of hardware SOMs. Moreover, we give an overview of main challenges and trends for their ubiquitous adoption as hardware accelerators in many application fields. This article is expected to be useful for researchers in the areas of artificial intelligence, hardware architecture, and system design.**

***Index Terms*— Application-specific integrated circuit (ASIC), data reduction, field-programmable gate array (FPGA), hardware, real time, self-organizing map, survey, vector quantization.**

TABLE I
ACRONYMS AND TERMS USED FREQUENTLY IN THIS ARTICLE

SOM	Self-Organizing map
HW	Hardware
SW	Software
ANN	Artificial Neural Network
PE	Processing Element
LUT	Look-Up Table
FPGA	Field Programmable Gate Array
CMOS	Complementary Metal-Oxide-Semiconductor
ASIC	Application Specific Integrated Circuit
GPU	Graphics Processing Unit
CPU	Central Processing Unit
SoC	System-on-chip
NoC	Network-on-a-chip
GWS	Global Winner Search
WTA	Winner-take-all
BMU	Best Matching Unit
neuron	SOM node
learning phase	Weights update through iterations
recall phase	BMU search in a learnt SOM map
prototype vector	SOM neuron's weight after training
codebook	SOM neuron weights after training (all prototype vectors)
learning kernel	neighbourhood function

I. INTRODUCTION

THE self-organizing map (SOM) is a special type of artificial neural network (ANN) proposed by Kohonen [1]. The SOM is an unsupervised learning algorithm performing a nonlinear mapping from a given high-dimensional input vector space to a low-dimensional map of neurons, usually a regular 2-D grid. It acts as a nonsupervised clustering algorithm as well as a powerful visualization tool, and it has been used to visualize, interpret, and classify large high-dimensional data in many application domains, such as economy, industry, management, sociology, geography, and text mining [2], [3].

Since its proposal, the implementation of the SOM algorithm has been the topic of many research works [4]–[41]. Indeed, different implementations of SOMs have been proposed so far: in software (SW) programmable platforms such as central processing unit (CPU) [29]–[33], [35]–[37] and graphics processing unit (GPU) [29]–[35] by using different programming languages (C/C++ w/wo OpenMP, Python, R, CUDA, and so on) or in dedicated hardware (HW) implemented on field-programmable gate arrays (FPGAs) or in application-specific integrated circuit (ASICs) by proposing specific hardware architectures exploiting the inherent parallelism of the SOM algorithm for better performances [4]–[28]. In applications requiring a large number of neurons (i.e., more than 10^2) and/or processing huge volumes of data (i.e., more than 10^4 of samples with dimensions $>10^2$), the SOM algorithm requires significant processing power that often cannot be provided with the conventional CPU-based computing platforms. In the last decade, as shown in Fig. 1, with a massive surge of general-purpose computation on GPUs (GPGPU), the SOM GPU-based implementations have gained an increasing interest due to their significant speedups with respect to CPU counterparts, as well as impressive improvements in terms of performances in comparison with HW SOMs [33]. On the other hand, the GPU processing power is to the detriment of the overall energy efficiency (number of operations, i.e., connection updates, per second per consumed watt), which is more than ten times higher for the FPGA-based SOMs, as reported in [35]. In addition, the substantial parallelism found in the SOM algorithm with the high energy efficiency is the main reason to target HW for SOM implementations. This article gives an overview of the hardware, application-specific implementations of the SOM algorithm, the most widely used computing blocks, architectures, design choices, adaptation,

Manuscript received June 30, 2021; revised November 3, 2021 and January 15, 2022; accepted February 14, 2022. This work was supported by JSPS KAKENHI under Grant JP20K11999. *(Corresponding author: Slaviša Jovanović.)*

Slaviša Jovanović is with CNRS, IJL, Université de Lorraine, 54000 Nancy, France (e-mail: slavisa.jovanovic@univ-lorraine.fr).

Hiroomi Hikawa is with the Department of Science and Engineering, Kansai University, Osaka 564-8680, Japan (e-mail: hikawa@kansai-u.ac.jp).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3152690.

Digital Object Identifier 10.1109/TNNLS.2022.3152690

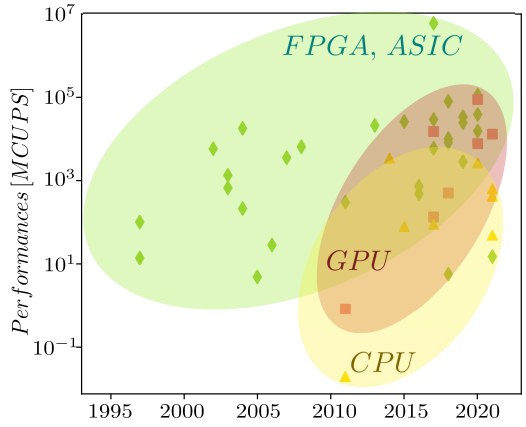

Fig. 1. Performance overview of different SOM implementations (in million of connection updates per second—MCUPS, see Section V-C) (1995–2021): HW (FPGA and ASIC) implementations (data extracted from [4]–[28], [35]); GPU implementations (data extracted from [29]–[35]); and CPU implementations (data extracted from [29]–[33], [35]–[37]).

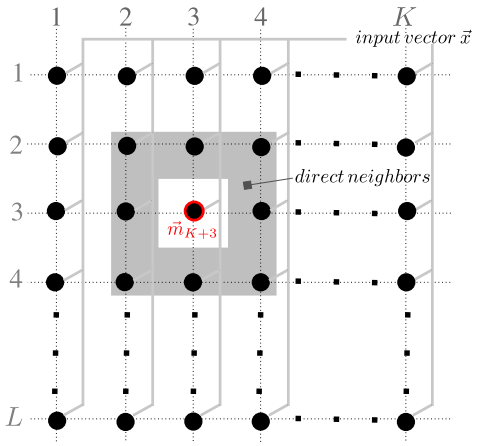

Fig. 2. Illustration of a 2-D SOM structure: the black nodes represent the neurons, the grayed area represents the direct neighbors of the neuron $K+3$, and the light gray lines represent the input vector delivery to all neurons.

and optimization techniques that have been reported in the literature so far in the field of HW SOMs.

The remainder of this article is organized as follows. Section II describes the original SOM algorithm, while Section III provides an overview of hardware adaptations of the original SOM algorithm in all phases (initialization, vector distance computation, BMU search operation, neighborhood function, and weight update). Section IV describes the types of architectures commonly found in hardware implementations of SOM. In Section V, the methods, tools, datasets, and application use cases for validation of hardware SOMs as well as an overview of their performance measurements are provided. Open research and challenging problems in the field of HW SOM implementations are discussed in Section VI. Finally, Section VII provides the final conclusion of the present survey. In addition, Table I shows acronyms and terms frequently used in this article.

Algorithm 1 SOM algorithm [1]

```
1  SOM Algorithm (G, X)
   inputs : The initial set G of neurons with weights m⃗;
            The set X of input vectors x⃗
   output: The set G of trained neurons
2  Initialization (G,parameters);
3  while Stopping criteria not met do
4      for λ random x⃗ ∈ X do
5          foreach i ∈ G do                          } Distance
6              d_i ← ||x⃗ − m⃗_i||_2                    } Recall
7          end                                       } Learning
8          c = argmin_{i∈G}(d_i)                     } BMU
9          if (learning) then
10             foreach i ∈ G do                      } Update
11                 ε_n ← h_ci(i, i_c)
12                 m⃗_i ← m⃗_i + ε_n(x⃗ − m⃗_n)
13             end
14         end
15     end
16 end
17 return G;
```

II. Background

A. Original Self-Organizing Map Algorithm

The original SOM algorithm proposed by Kohonen is summarized in Algorithm 1. Its starting point is a map of neurons $i \in G$ usually placed in a two-dimensional $L \times K$ grid, as shown in Fig. 2. Every neuron $i \in G$ on the map includes a D-dimensional vector $\vec{m}_i \in \Re^D$, called the weight vector

$$\vec{m}_i = \{\mu_{i,0}, \mu_{i,1}, \ldots, \mu_{i,j}, \ldots, \mu_{i,D-1}\} \in \Re^D. \quad (1)$$

The learning phase starts with an appropriate initialization, where each weight vector's element $\mu_{i,j}$ ($0 < i \leq L \cdot K, 0 \leq j < D$) is initialized with a random value. In the learning phase, which is carried out through $\lambda \in \mathbb{N}$ steps or iterations, the map is trained with a set of training vectors $\vec{x} \in X \subset \Re^D$

$$\vec{x} = \{\xi_0, \xi_1, \ldots, \xi_j, \ldots, \xi_{D-1}\} \in \Re^D. \quad (2)$$

At the beginning of each iteration, a training vector $\vec{x}$ is delivered to all neurons of the map, as shown with light gray lines in Fig. 2. In each iteration, all neurons calculate the distances of their weight vectors $\vec{m}_i$ with respect to the input vector $\vec{x}$. Then, the neuron-$C \in G$ that has the closest weight vector $\vec{m}_C$ to the input vector $\vec{x}$ is determined from the calculated vector distances of all neurons

$$C = \arg\min_i \{\|\vec{x} - \vec{m}_i\|_2\}. \quad (3)$$

This search for the neuron having the shortest vector distance is often called a winner-take-all (WTA) or best matching unit (BMU) operation, while the elected neuron is called winner or BMU neuron.

In Kohonen's study [1], the Euclidean metric is used as the vector distance to find similarities between input $\vec{x}$ and weight vectors $\vec{m}_i$

$$\|\vec{x} - \vec{m}_i\|_2 = \sqrt{(\xi_0 - \mu_{i,0})^2 + \cdots + (\xi_{D-1} - \mu_{i,D-1})^2}. \quad (4)$$

After the winner neuron is determined, the weight vectors of the neurons in its neighborhood are updated toward the input vector as

$$\vec{m}_i(t+1) = \vec{m}_i(t) + h_{ci}\left[\vec{x}(t) - \vec{m}_i(t)\right] \quad (5)$$

TABLE II
TIME COMPLEXITY OF THE MAIN OPERATIONS OF THE SOM ALGORITHM [42]

	Operations	Sequential	Parallel	
			Ideal	In practice[a]
Compute	Distance	$\mathcal{O}(n^b D^c)$	O(D)	O(nD/P[d])
	BMU	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log P)^e$
	Update	$\mathcal{O}(nD)$	$\mathcal{O}(D)$	$\mathcal{O}(nD/P)$
	Neighbourhood	$\mathcal{O}(nD)$	$\mathcal{O}(1)$	$\mathcal{O}(n/P)$
Data	Supply $\vec{x}$	$\mathcal{O}(nD)$	$\mathcal{O}(1)$	$\mathcal{O}(D)$
	Broadcast BMU	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
	Total	$\mathcal{O}(nD)$	$\mathcal{O}(D+n)$	$\mathcal{O}(\log P + nD/P)$

[a] Best values
[b] $n = LK$ - the total number of neurons in an $L \times K$ map
[c] D - the input vector dimension
[d] P - the number of processing units used for computation
[e] Global BMU operation; the associated local BMU with $\mathcal{O}(n/P)$

where $t \in \mathbb{N}$ represents the discrete-time coordinate and h_{ci} is the neighborhood function used to find the neighborhood neurons in the vicinity of the winner neuron whose weights are updated at the end of an iteration. Originally, the neighborhood function h_{ci} is defined as follows [1]:

$$h_{ci} = \alpha(t) \exp\left(-\frac{||\vec{r_C} - \vec{r_i}||}{2\sigma^2(t)}\right) \tag{6}$$

where $\vec{r_C} \in \Re^2$ and $\vec{r_i} \in \Re^2$ are position vectors of the winner neuron-C and neuron-i, respectively, and $\alpha(t)$ and $\sigma(t)$ are learning rate and neighborhood radius, respectively.

After the unsupervised learning with the training vectors, the SOM builds the weight map representing the quantized projection of the input vector space, whose probability density function is represented with the distributed prototype vectors of the SOM weight map. Moreover, the weights of the SOM are retained and are used in the recall phase, where only the winner neuron search is carried out without weight update (see Algorithm 1). An example of the behavior of a 16×16 SOM processing three-dimensional vectors during the learning phase is shown in Fig. 3. Small red dots distributed in a triangular shape represent the input vectors, and the larger blue plots specify weights of neurons, while lines connect the weights of nearest neighbors' neurons. Since the weight vectors are initialized with small values, they cluster at the origin of the plot, as shown in Fig. 3(a). Weights then gradually expand in an orderly way through numerous training iterations (E represents training iterations) until they approximate the distribution of the input samples. Note that the positional relationship between neurons in the vector space is maintained during training. This is possible due to the SOM inherent topology-preserving nature, which maps input vectors close to each other in the input vector space onto neighboring neurons of the SOM.

In the SOM algorithm, during an iteration, each neuron carries out the vector distance computation and the weight vector update. These operations are independent between neurons and can be performed simultaneously (see Algorithm 1). Consequently, this inherent parallelism of the SOM algorithm makes it suitable for hardware implementations. Neurons implemented in hardware can work in parallel, while the neurons in software are executed sequentially. Therefore, the hardware-implemented SOM can process its input vectors more efficiently than the software implemented one. This efficient computational power is especially desired when the size of the SOM or the input vector dimension is large. Due to the parallelism of the algorithm, the more neurons or vector processing elements (PEs) are implemented in hardware, the higher performance is obtained. In addition, the number of neurons implemented in hardware can be increased easily because they work independently of other neurons. As a result, the hardware SOM has the potential to provide high scalability. However, the bottleneck of the SOM algorithm is the winner search operation, in which the shortest vector distance must be searched. Indeed, all vector distance values must be compared to find the shortest one, and various hardware comparison methods have been proposed so far, as discussed in Section IV. Moreover, Table II summarizes the time complexity of the main operations of the SOM algorithm, for sequential and parallel implementations as well [42] supports the idea that by introducing SOM operation-specific hardware, a considerable gain in overall performances can be obtained.

III. SOM ALGORITHM FOR HARDWARE IMPLEMENTATON

As mentioned earlier, the SOM algorithm is suitable for hardware implementation because of its inherent parallelism. However, hardware expensive functions, such as Gaussian function in the original SOM algorithm, makes the hardware implementation challenging because the hardware resources of implementation platforms are limited. Therefore, the SOM algorithm has been further modified to be more suitable for hardware implementations.

This section provides an overview of hardware adaptations of the original SOM algorithm in all phases: initialization, vector distance computation, BMU search operation, neighborhood function, and weight update (see Table III for the overall summary).

A. Initialization

Even though, in the original SOM algorithm, it is suggested to use random initial values for demonstrations purposes, it turns out that this initialization policy does not necessarily provide the best performances in terms of convergence speed to the stationary values and quality of resulting maps [1], [43]. The initialization of SOMs has been widely studied in the literature [1], [43]–[48]. Basically, two groups for initialization purposes can be found [1], [44], [45]: random initialization and linear initialization (often called data-driven initialization). In the random initialization, the weights of the SOM neurons are initialized: either by randomly choosing data in the range of values observed in the input dataset, by randomly selecting the data from the input datasets, or by randomly choosing the perturbed values around the mean values observed in the input dataset. In the data-driven or linear initialization, the input datasets are previously analyzed before initializing the SOM weights. Different methods can be found in the literature: based on the data analysis with k-means where input datasets are projected on the map of the same size in order to find the cluster centers that are then rearranged with some

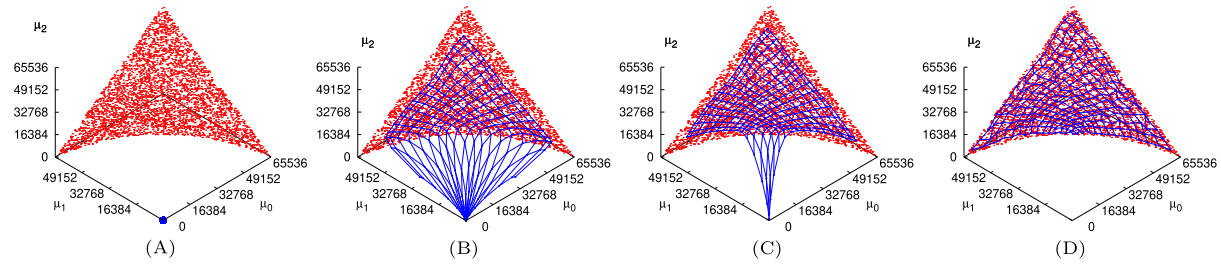

Fig. 3. Weight vectors during training. (a) $E = 0$. (b) $E = 1$. (c) $E = 3$. (d) $E = 256$.

heuristics to fit the used SOM map [44]; on the statistical analysis such as principal component analysis (PCA) on regular grids [47] where largest eigenvectors of the projected input dataset are chosen; or the same PCA-based analysis with projection on irregular grids [45], by using connected graphs to rearrange the most distant elements in the input dataset of nonvectorial elements [46], or by placing the SOM neurons on the Hilbert self-similar curves [47]. Akinduko *et al.* [48] compared random and PCA-based initialization on quasi-linear and nonlinear datasets. They concluded that there is no universal initialization policy giving the best performances in terms of convergence and quality of obtained results; the random initialization performs better on nonlinear datasets, whereas the PCA-based initialization gives better results for quasi-linear ones.

When the SOMs are implemented in software, as it was the case in all initialization techniques previously presented, the weights can be easily programmed. However, the vector initialization is a burden for the hardware implementation since it requires additional circuits such as a linear feedback shift register (LFSR) that provides random values. If a single LFSR is employed, all vector elements must be initialized one by one, and thus, a communication link between the initialization circuit and all neurons is necessary. On the other hand, each neuron can have its own LFSR. In this case, the global communication link is not necessary, but all LFSRs must be differently initialized to generate different values, which thus breaks modularity of neurons. A weight programmability is necessary for neurons to modify their weights, but additional circuits to initialize all weight vector elements increase the hardware cost of the neurons. Thus, if possible, it is desirable to omit the initialization of the weight vector.

Kolasa *et al.* [49] investigated the weight initialization problem of the hardware SOM by simulations. Comprehensive simulations were carried out for several initialization strategies with different scenarios, and the result revealed that the hardware SOMs in many situations could be trained without any initialization, simply by using zeroed weights. These results are explained by the influence of the SOM algorithm's neighborhood mechanism that in a given learning cycle stimulates also the activity of the neurons topologically positioned in a broader vicinity of the winner neuron.

This study was followed by [50] where an efficient initialization of HW SOM neuron weights was investigated. The obtained results confirmed that the SOM could learn properly even if the learning process started with zeroed weights. In addition, in this work, an initialization circuit with full programmability of the weights was also proposed for the map without the neighborhood function.

B. Vector Distance Computation

The Euclidean metric is one particular type of a Minkowski metric that can be considered as a generalization of both the Euclidean distance and the Manhattan distance. The Minkowski distance is defined as

$$D(\vec{x}, \vec{m}) = \left(\sum_{i=0}^{D-1} | \xi_i - \mu_i |^L \right)^{\frac{1}{L}}. \tag{7}$$

By setting $L = 2$, the Minkowski distance corresponds to the Euclidean distance. The Minkowski norm with $L = 1$ is known as Manhattan, City block, or Taxicab metric

$$D_M(\vec{x}, \vec{m}) = \sum_{i=0}^{D-1} | \xi_i - \mu_i | . \tag{8}$$

Since no squaring or square root circuit is required, the silicon area saving caused by the Manhattan metric is significant, and it is well suited for hardware implementation. Dlugosz *et al.* [51] investigated the effect of the Euclidean and Manhattan distances on the learning. The detailed system-level simulations showed that the Euclidean and Manhattan distances both lead to similar learning results. Another variant of the Minkowski distance is the Chebyshev distance (also known as chessboard distance) that can be obtained with $L = \infty$

$$\lim_{L \to \infty} \left(\sum_{i=0}^{D-1} | \xi_i - \mu_i |^L \right)^{\frac{1}{L}} = \max_{i=0}^{D-1} | \xi_i - \mu_i | . \tag{9}$$

Pena *et al.* [11] investigated the use of the Chebyshev distance in HW SOMs and concluded from FPGA synthesis results that the Manhattan distance outperforms the Chebyshev one in terms of speed but at the expense of higher HW resources (bigger chip area).

In the original SOM algorithm, the computed Euclidean distances of all neurons are compared to find the shortest one. Even if the square root is not calculated, there is no effect on the magnitude comparison of the distances. Therefore, another popular vector distance metric is squared Euclidean distance because the hardware cost of the square root function is very expensive

$$D_S(\vec{x}, \vec{m}) = \sum_{i=0}^{D-1} (\xi_i - \mu_i)^2. \tag{10}$$

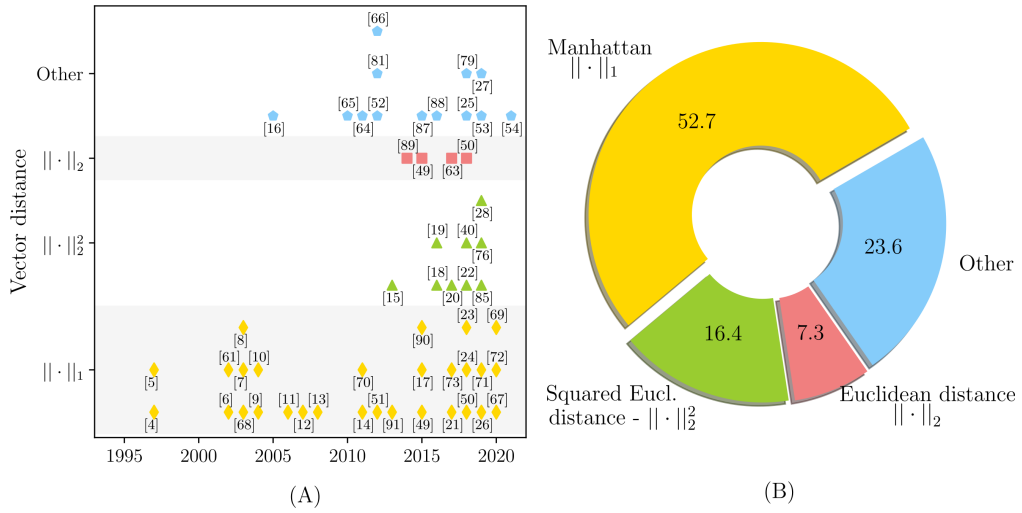

Fig. 4. Vector distance functions in HW SOM implementations (1995–2021): (a) overview and (b) percentage chart.

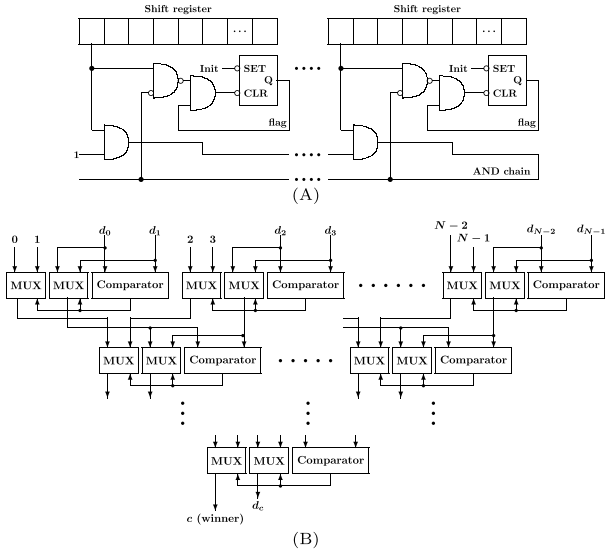

Fig. 5. BMU search circuit. (a) Bit serial. (b) Binary tree.

The types of the vector distance that have been used in the hardware SOMs so far are summarized in Fig. 4 and Table III. It can be noticed that more than 75% of the used vector distances in the HW SOMs in the last 25 years (1995–2021) are the Manhattan, squared Euclidean, and Euclidean distance. This ongoing trend is also found in the recent state-of-the-art HW SOM implementations. The last quarter ("Other" in Fig. 4) is the different attempt to use uncommon distance metrics for the HW SOM algorithm: modified Hamming distance [52], frequency comparison [25], dot product cosine [53], inner product [27], count of cycle slips [54], and so on.

C. BMU (Winner) Search

In the winner search, often referred to as the BMU or WTA search operation, the neuron unit that has the closest weight to the input vector (shortest distance) is searched. In HW SOMs, this operation is implemented either with analog or digital BMU circuits.

1) Analog BMU Search Circuit: Different analog hardware WTA computation circuits have been proposed in the literature. One of the WTA circuits is MAXNET [55], where neurons in the network mutually inhibit each other while activating themselves. Eventually, only one neuron is kept activated and becomes the winner. Lazzaro *et al.* [56] proposed a CMOS WTA circuit, where signals are represented as analog currents. Oster*et al.* [57]examined analytically the ability of a spike-based WTA network. Similarly, other examples of spiking WTAs with temporal coding have been reported in [58]–[60].

2) Digital BMU Search Circuit: Among digitally implemented WTAs, a bit-serial parallel minimum search circuit, shown in Fig. 5(a), is reported in [1]. The bit-serial winner determination is based on a bit-by-bit comparison of all neurons' vector distances, performed in parallel. This calculation requires a global AND operation between all neurons and feedback to them. The distance of each neuron is loaded to a shift register and its corresponding flag is set to "1" at the beginning of the search operation. All the neurons in the network, in particular the most significant bits (MSBs) of their shift registers, are sequentially connected through AND gates. The AND chain signal becomes low if at least one of these MSB bits is zero. The bit comparison starts from the MSB and proceeds through the least significant bit (LSB) by shifting the register to the left, covering one bit per stage (per clock). At any stage, the flags of the neurons with MSB = "1" are reset to zero if one of the MSBs of all the neurons is zero (the AND chain signal is zero). Otherwise, the flag is unchanged if it is already zero. Resetting the flag of a neuron to zero eliminates that neuron from further competition. After the last stage, only the neurons with shortest distances are left and their flag is "1." Therefore, this method takes L clock periods to complete the search, where L is the bit length of the distance norm. Similarly, Tamukoh *et al.* [9] proposed a WTA circuit, which is a modified version of the

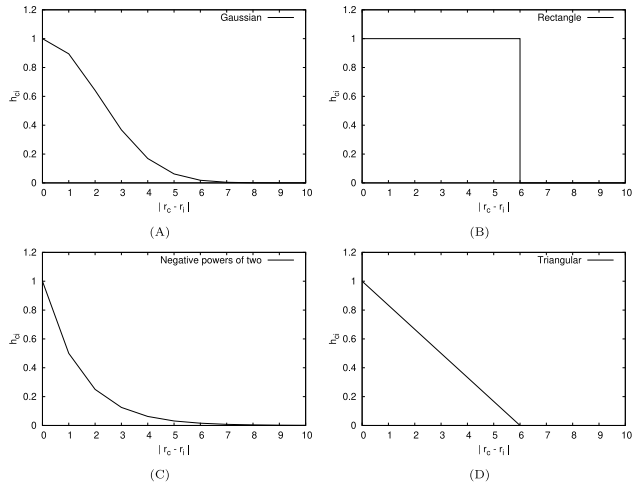

Fig. 6. Neighborhood functions. (a) Gaussian, (b) rectangular, (c) negative powers of two, and (d) triangular.

bit-serial comparison. The proposed WTA circuit performs a rough comparison of neurons' distances in the early stage and a strict one later, which allows the faster learning in massively parallel SOM architectures.

Hikawa *et al*. [23] modified the bit-serial comparison circuit so that all neurons' distance bits at the same position are tested simultaneously without any shift registers. Thus, most significant bits of the neurons' distance are excluded from the winner search. The comparison results indicating whether a neuron is still a candidate for the winner or not are propagated to the lower bit through a signal instead of being stored in a flag register, as it was the case in the bit-serial winner search. At the LSB position, the neuron having the shortest distance is determined. This winner search circuit is a pure combinatorial circuit (no clock), leading though to a longer latency (poorer performance) of the circuit.

Another popular method of doing fast BMU search is using several parallel comparators in a binary tree structure. An example of the binary tree search circuit is shown in Fig. 5(b). Each comparator is accompanied with two 2-to-1 multiplexers (MUX) whose selection signals are controlled by the comparator's output. The winner is selected by tournament selections, and the multiplexer forwards the shortest distance and the corresponding neuron ID to the next stage. This parallel binary-tree BMU search method is implemented as a global WTA circuit that collects vector distance data from all the neurons of the map.

Mailachalam *et al.* [61] compared the bit-serial minimum and the binary-tree search circuits in terms of the speed, hardware resources (chip area), and circuit complexities. They concluded that the binary tree method is faster than the bit-serial method, allowing to build HW SOMs with better performances. On the other hand, in terms of needed hardware resources for their implementation, often presented with the chip area the designed circuit will have once manufactured, no difference was found.

The hardware SOM proposed by Hendry *et al*. [7] employs a unique winner search circuit. In this method, the distance value stored within each neuron is decreased by one at each clock cycle until zero is reached. The neuron or neurons, which first reach zero, candidate themselves to become the winner by attempting to output their identifier to the global output bus. Since it is possible that several neurons have the same minimum distance, the global winner is randomly chosen from those neurons.

Kurdthongmee [62] proposed a low-latency digital BMU search circuit for HW SOM quantizers. The general idea is to use a K-address 1-bit memory, where K corresponds to the maximal value of distance encountered in a given application (here color quantization). The 1-bit memory location is used to indicate whether the corresponding distance is found ("1" or "occupied") or not ("0" or "unoccupied") in a given learning iteration. At the end, the first "occupied" memory location starting from the beginning designates the BMU for a given learning iteration. The index of the first "occupied" memory location is retrieved with a custom lookup table (LUT)-based circuit. The proposed BMU search approach reduces the latency of the overall BMU operation (0.62 times of the conventional method with comparators and binary tree) to the detriment of the hardware cost (2× overhead for an FPGA implementation).

D. Neighborhood Function

A very important feature of the SOM is its topology-preserving nature, where two adjacent vectors in the input vector space are mapped onto adjacent neurons on the map. This topology-preserving nature is realized by the weight update with the neighborhood function, which affects as well the performance of the SOM in the recall phase. The original algorithm uses the Gaussian neighborhood function shown in (6) and Fig. 6(a). However, the Gaussian neighborhood function is not suitable for hardware implementation because of its high hardware cost. Straightforward way to implement the Gaussian function is to store precomputed values in a lookup table (LUT). The size of neighborhood depends on the SOM size, and thus, the memory content that implements the LUT-based Gaussian function tends to be large, leading to the need for more hardware resources. Moreover, the LUT-based neighborhood function requires the use of multiplier for vector adaptation, which should be avoided due to its higher hardware cost. Thus, most of the hardware SOMs use simplified neighborhood functions.

The most simple neighborhood function is a rectangular or step function, as shown in Fig. 6(b). With this function, only weight vectors of the neurons within a certain radius R from the winner neuron are updated with the same update coefficient α. Since no multiplication is necessary to implement the neighborhood function, the hardware cost for the neighborhood function is reduced. The works in [5], [7], [15], [52], and [63] used the rectangular function.

A function, the value of which is restricted to negative powers of two, is another popular neighborhood function in HW SOMs. The multiplication can be replaced by a shift operation because the values are restricted to the negative powers of two. This function is shown in Fig. 6(c). The

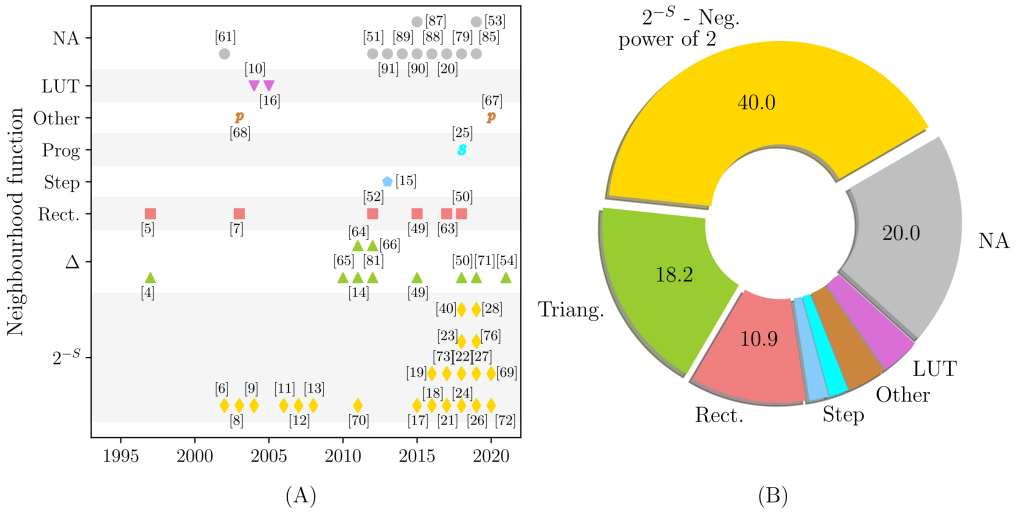

Fig. 7. Neighborhood functions in HW SOM implementations (1995–2021): (a) overview and (b) percentage chart.

multiplication by the neighborhood function is executed by the right-shift arithmetic operation, and its shift size is determined by the distance to the winner neuron. As shown in Fig. 7 and Table III, many hardware SOMs employ the negative-powers-of-two neighborhood function.

Dlugosz *et al.* [14], [64] proposed a triangular neighborhood function. As shown in Fig. 6(d), the function value linearly decreases with the distance to the winner neuron. The effect of this triangular neighborhood function on the SOM learning was studied by Hspice simulation [65], [66] and revealed that it can be successfully used instead of the Gaussian neighborhood function. Moreover, the sound performance of the SOM was also confirmed in the cases where the signal resolution of the neighborhood function was low. However, the triangular neighborhood function requires multipliers for its computation, leading to a higher hardware cost.

The hardware SOM proposed by Pohl *et al.* [10] is based on the modular array architecture where neurons exchange data via a shared bus. In this approach, during each iteration, the distance value to the winner in every neuron is decreased by one at each clock cycle. Then, the neuron whose distance reaches zero takes a value from the shared data bus, which represents the update coefficient for the neighborhood function. These values, provided by the bus to all the neurons of the architecture, are the neighborhood function values previously precalculated in software and stored in the memory.

The neighborhood function proposed in [16] is realized as a combination of an update pulse generator and an update pulse selector. The update pulse generator generates three pulse signals with different frequencies. The amount of update to use by the neighborhood function is proportional to the chosen frequency. Thus, the winner neuron uses the highest frequency update pulse, whereas its neighboring neurons use the lower frequency ones. To provide satisfactory learning results, these frequencies are determined and predefined offline. The neighborhood mechanism used in [17] was implemented in the similar way, where three precalculated neighborhood function values are broadcast to all the neurons of the map. Each neuron uses the appropriate value according to its distance to the winner neuron. In addition, the function values are made adaptively large to cover the cases where the distance to the winner neuron is larger than the usual one. Thus, the learning performance in terms of quantization error is also improved.

Despite the optimizations mentioned above, all hardware implementations of the SOM presented in the literature are affected by a common problem, i.e., the performance decreases with an increasing number of neurons. The winner search operation, commonly realized with a comparator tree, is the main reason for this performance degradation (see Section III-C and Table II). Cardarilli *et al.* [67] proposed the all winner-SOM (AW-SOM), a modified version of the SOM algorithm. The AW-SOM is based on the following idea; if the input vector is close enough to the winner neuron, the coordinates of the former can be used directly in the neighborhood function. Because the AW-SOM does not use the conventional neighborhood function that requires the coordinate of the global winner, it does not require neither the identification of the winner neuron, which is the most critical operation in terms of propagation delay. Experimental results show that if the neurons are initialized using a uniform or random distribution, the results of AW-SOM and traditional SOM clustering are comparable in 92% of the cases. The failed cases are related to the bad position of the clusters with respect to the initial position of the neuron. In addition, the absence of the comparator tree for the winner neuron selection considerably improves the system performance (see Section V-C). On the other hand, because the topological relations between neurons are not considered and the winner-based neighborhood function is not used as well, the topology-preserving nature of the SOM, stated earlier as one of the pillars of the SOM algorithm, is not guaranteed in this approach.

The types of the neighborhood functions that are used in the hardware SOMs are summarized in Fig. 7 and Table III. It can be noticed that almost a half of the used neighborhood functions in the HW SOMs in the last 25 years (1995–2021) is

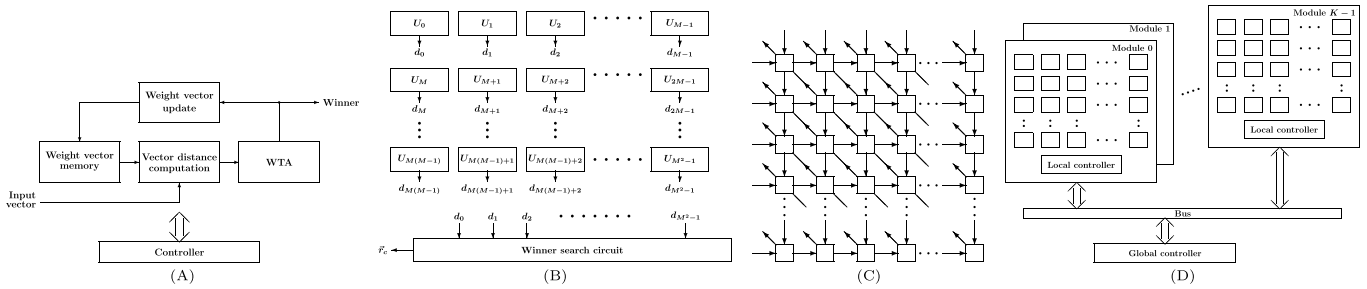

Fig. 8. Hardware SOM architecture types. (a) Dedicated processor, (b) distributed, (c) systolic array, and (d) modular.

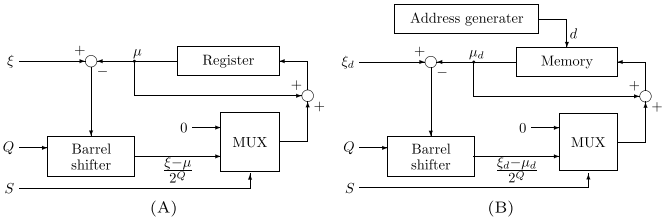

Fig. 9. Weight update methods with (a) register and (b) memory.

the negative-powers-of-two function, which is also the current trend in the recent state-of-the-art HW SOM implementations, followed by triangular (16.7%) and rectangular neighboring function (11.1%). Some examples of LUT-based [10], [16] or programmed [25] neighboring functions can also be found as well as the HW SOMs without neighboring function [67], [68].

E. Weight Vector Update

Hardware SOMs handling low-dimensional vectors, such as [11], [24], [26], use registers for storing the weight vector elements. Fig. 9(a) shows an example of weight update circuit for a single-element vector. This circuit comprises a register, where the weight value is stored, a barrel shifter for the neighborhood function, and a 2-to-1 multiplexer to choose if the weight value needs to be updated or not. The circuit first calculates $(\xi - \mu)$, which is the difference between the input and the weight vector element. This difference is then fed to a barrel shifter that outputs $(\xi - \mu)/2^Q$, which is a Q-bit shifted value to the right of the initial difference. If the signal S is one, the barrel shifter output is added to the register; otherwise, no change in the weight value. The values of S and Q are determined by computing the distance between the neuron whose weights are updating and the winner one. Consequently, by controlling S and Q, the negative-powers-of-two neighborhood function is implemented. In addition, to process D-dimensional vectors, D update circuits are employed to update D vector elements simultaneously.

On the other hand, in the HW SOMs targeting higher dimensional vectors, such as [9], [19], [69], a memory block is often employed to store the weight vector elements. In this case, a typical update circuit is shown in Fig. 9(b). It can be noticed that is heavily inspired by the update circuit used for low-dimensional vectors and presented in Fig. 9(a). The main difference is the replacement of the register with a memory block and the addition of an address generator, whose function is to provide memory addresses to access all weight vector elements. In this update circuit, the weight vector elements are read out sequentially from the memory and are modified in the same way as in Fig. 9(a).

IV. Hardware SOM Architecture

As substantial parallelism is found in the SOM algorithm, various hardware SOM architectures aiming to speed up its computation and provide better both, learning and recall performances, have been proposed in the literature. These hardware SOMs, which employ a high degree of parallel computation to accelerate the SOM algorithm, are listed in Fig. 10 and Table III. Typically, four types of SOM computing architecture, shown in Fig. 8, can be found in the HW SOMs (see Fig. 10 for percentage chart):

1) dedicated processor;
2) systolic array;
3) distributed;
4) modular architecture.

A. Dedicated Processor

The dedicated processor hardware, shown in Fig. 8(a), typically consists of computing components dedicated for the SOM computation. These components are a memory for storing all weight vectors, a vector distance computing unit, a WTA unit for the winner search operation, and a component performing the weight update. Since the SOM computation is based on the vector arithmetics, the computing units in this processor (the vector distance and weight update units) can handle vectors.

Asanovic [4] implemented the SOM algorithm within the Spert-II system based on a T0 vector microprocessor. The T0 microprocessor includes an MIPS-II compatible RISC CPU with a 1-kB on-chip instruction cache, a fixed-point vector coprocessor, and an external memory interface. By making use of these available parallel computation resources, the SOM computation was highly accelerated (more than $10\times$ compared to the state-of-the-art architectures at the time).

Kurdthongmee [70] proposed a color image compression system based on a hardware SOM, which employs a rational-numbering system for the codebook and learning kernel. Use of the rational numbering system and approximated nonlinear learning kernel extends the capabilities of the quantizer. The experimental results proved that the quality of the

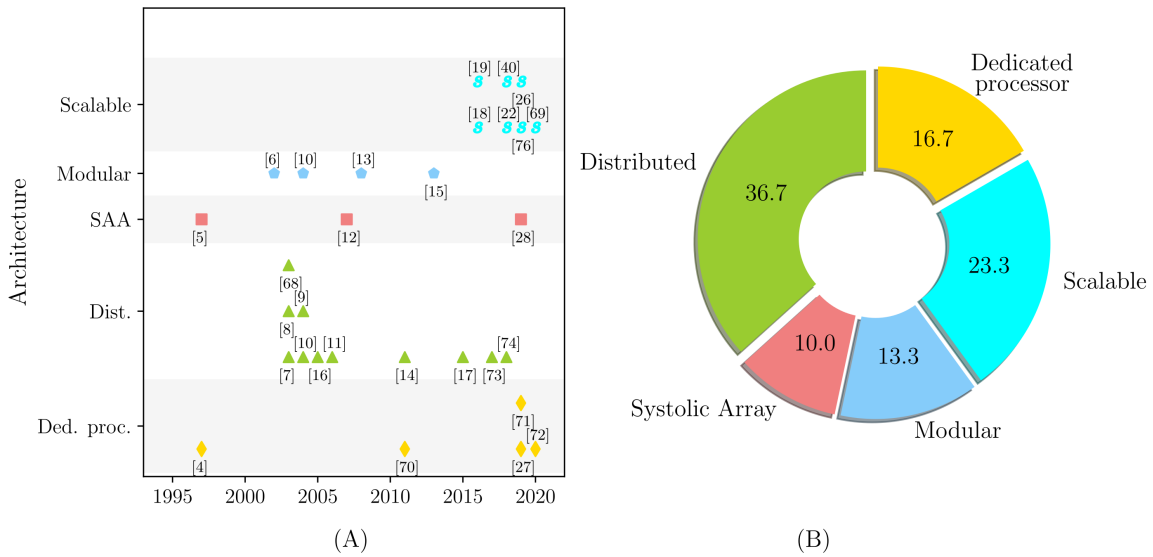

Fig. 10. Architectures in HW SOM implementations (1995–2021): (a) overview and (b) percentage chart.

outcome images is superior to predecessor implementations with an acceptable throughput and FPGA resource utilizations.

Tanaka *et al.* [27] developed a single layer of a deep SOM network and a fully connected neural network (FCNN), used to mimic the function and structure of the amygdala. The amygdala is a specific area of the brain associated with classical fear conditioning. In general, the performance of deep neural networks (DNNs) is quite reliant on the availability of large amounts of training data, which is not always present. The authors tackled this problem by the brain-inspired amygdala model to achieve computer learning with limited training data. Hardware for the deep SOM network and the FCNN was designed and implemented on an FPGA, and the proposed amygdala model was applied to a robot waiter task in a restaurant. The experimental results show that the model learns a customer's preferences after only a few human–robot interactions.

Sun *et al.* [71] presented a hardware platform for accelerating the SOM algorithm. In the proposed platform, four types of neuron network topology are supported: 1-D array, 2-D square, 3-D cube, and binary tree. The proposed acceleration circuit contains eight processing units performing weight update and distance calculations for eight neurons simultaneously. It also includes a comparator tree to find the winner neuron. The proposed accelerator was applied to three applications: chromaticity diagram learning, labeling of handwritten number image, and image vector quantization. The functionality of the proposed platform was proven by MATLAB simulations of these applications, whereas the hardware cost estimation was carried out through the FPGA synthesis.

de Sousa *et al.* [72] proposed an FPGA-based SOM architecture called SOMprocessor. The proposed architecture explores two different computational strategies to improve both the data flow through the processor and its flexibility to implement different network topologies. The first improvement is achieved by multiplexer components, which supports alternating processing of neuron sets by the arithmetic circuits. This strategy provides a more flexible use of the chip, in which larger networks can be even processed in low-density FPGAs. The second improvement is the inclusion of a pipeline architecture for the training algorithm so that different parts of the circuit could process data at the same time. The SOMprocessor was applied to a video categorization task, on the example of human actions video categorization for autonomous surveillance.

B. Distributed Architecture

The distributed architecture shown in Fig. 8(b) is based on the use of multiple PEs. Thus, it can be regarded as a dedicated array processor for the SOM architecture. Also, it is considered as a massively parallel architecture when a large number of PEs are included in the array. All PEs perform a part of the SOM computation in parallel. Global computations, such as the winner neuron search, are carried out by the PEs, whereas interconnections propagate the necessary information throughout the entire structure. Each PE may process multiple neurons and the highest parallelism is achieved if each PE processes a single neuron, which also yields the highest efficiency.

1) Processing Element: The PEs execute the same computation on different data, i.e., weight vectors. Thus, some researchers, such as [7], [8], [10], employed a single instruction multiple data (SIMD) processor for flexibility. The same instructions or commands are generated by a global controller and fed to the SIMD processors. Hendry *et al.* [7] presented their SIMD array for the hardware SOM as a soft IP core so that the number of neurons, the number of elements per vector, and the number of bits of each element could be defined by synthesis time parameters.

Another example is the hardware SOM proposed by Sudha *et al.* [68]. They presented a novel hardware architecture for a 3-D SOM, designed for color quantization. Color quantization is a process of generating a color palette containing a

limited number of colors from a full-color image and can be done with the SOM. The generated color palette is afterward used to reconstruct the image (i.e., for image compression purpose). Each pixel of the reconstructed image is replaced with one of the colors from the palette. This process allows to quantize the initial color of each pixel in the image and thus reduces the image data size without significantly degrading the image quality. In the 3-D SOM, the neurons are arranged as points on a cube with their red, green, and blue axes. The sequence of operations in the coding phase is controlled by the instructions generated and propagated by the control unit of the proposed SOM architecture.

Another approach to build SOM PE is by using dedicated PE circuits made of hardwired logic circuit defined at the register transfer level (RTL). Since this approach does not have to decode the instructions or commands, the higher performance is yielded at the expense of the overall flexibility. This type of hardware neurons was used in the hardware SOMs listed in Table III and labeled as "distributed" in the "architecture type" column.

2) Interconnection Links: The distributed architecture requires communication links between the PEs, the controller, and I/O. Examples of a global bus-based communication method can be found in [9], [16], [17]. In these approaches, a global winner search (GWS) operation is commonly carried out. The GWS circuit collects all weight vector distances from all neurons, and the global winner is searched and identified.

The works, such as [11], [14], [73], employed the local interconnection network, in which the PEs are connected with their direct adjacent neurons. The input vectors are fed to the SOM through the local links and the winner neuron search is distributed among groups of neurons by using the local interconnections. The local winner within a group of direct neighboring neurons (next to each other) is searched first. Then, the global winner is determined by comparing the local winners from all the neuron groups.

de Abreu de Sousa *et al*. [21] compared three types of architecture for executing the SOM learning and recall phases: distributed, centralized, and hybrid. The centralized architecture is defined as the architecture using a central control unit to collect the distance information from all neurons and for the BMU search as well. In the distributed architecture, the local winner of the neighboring neurons is computed and broadcast to all other neurons. This process is continuously repeated, and at the end, the global shortest distance value is propagated through the entire network. The third architecture is a hybrid model, which is a mix of the two previous architectures. The FPGA implementations of the three models were compared to support the system design choices. Results show that the centralized model outperforms the other models in terms of chip area occupation and maximum operating clock frequency.

Rodriguez *et al*. [74] presented the generic iterative grid principles for distributed computing. The use of the iterative grid to implement three types of SOMs, i.e., the original Kohonen SOM (KSOM), the dynamic SOM (DSOM), and the pruning cellular SOM (PCSOM), was investigated. Due to the iterative grid, the implementations of those SOMs are fully decentralized. The behavior of these iterative grid models was simulated and was found to be competitive to centralized models using the Manhattan distance as the vector distance metric.

C. Systolic Array Architecture

Based on the distributed PE array structure, previously introduced, some of the HW SOM architectures, such as the works presented in [5], [12], and [28], employ also the systolic model of data exchange. For instance, the systolic array proposed by Kung [75] is a homogeneous network of tightly coupled PEs, as shown in Fig. 8(c). Each PE independently computes a partial result using the data received from its neighborhood PEs in the upstream direction(s), stores the result within itself, and passes it to its neighboring PEs in the downstream direction(s). Thus, every PE performs different computation. In addition, the data path is aggressively pipelined allowing to increase the overall architecture performances.

Ienne *et al*. [5] employed the MANTRA I system to validate the new SOM learning algorithm they proposed. The MANTRA I system is a massively parallel system based on the systolic array of up to 1600 PEs. The systolic array at the heart of the SIMD part of the proposed architecture is a square mesh of GENES IV PEs, designed in CMOS 1 μm standard-cell technology. In the original learning algorithm, the weights are updated after the presentation of every input vector. On the contrary, in the proposed architecture, the weight vectors are updated after the presentation of a group of inputs in a batch fashion. Consequently, the authors proposed a new weight update algorithm called mantra algorithm, which is an intermediate between the original and the batch algorithm. With the mantra algorithm, the winner selection is based on the batch algorithm, and the weights are updated by using the original method. It was validated through theory and simulations that the mantra algorithm performs almost as well as the original learning algorithm. The main advantage of the mantra algorithm is its finer grain of parallelism, which allows it to be used in hardware with a very large number of processors without compromising the properties of the algorithm.

Manolakos *et al*. [12] designed a modular SOM systolic architecture, described as a soft IP core in a synthesizable VHDL. The network size, the vectors dimension, the weight and data element bit width precision, and so on are all tunable parameters. The proposed array is made of two types of PEs: the recall mode PE (PER) and the weights update PE (PEU). An SOM module control unit (MCU) generates all the necessary control signals for both array columns. The proposed SOM was implemented on an FPGA and validated on a vector classification task in real time working on input vectors with thousands of elements.

Similarly, Ben Khalifa *et al*. [28] proposed a new SOM architecture called systolic-SOM (SSOM). The SSOM is based on the use of a generic model, also inspired by a systolic movement. In this model, two levels of nested parallelism of neurons and connections are used. The proposed approach was validated and its performances were evaluated through several different SOM networks integrated on an FPGA platform.

D. Modular Architecture

Problem of the distributed architecture is its expandability. In order to increase the number of neurons, the whole system must be redesigned. The modular architecture shown in Fig. 8(d) divides the whole SOM hardware into modules. Each module is usually the distributed architecture SOM (see Section IV-B), and the number of neurons can be increased easily by adding new modules. Thus, this approach provides the expandability to the HW SOM architecture.

Lachmair *et al.* [15] proposed hardware SOM based on the modular architecture, called gNBXe. This work is based on the principles of NBX and gNBX, previously introduced in [6] and [10], respectively. It consists of a global controller (GC) and the PEs that are the hardware units performing the calculations for simulating the neurons. The local controller in a PE translates the macro commands from the global controller to the local control signals, allowing the corresponding PE to perform the calculations related to the neuron specified in the macro command. Consequently, the system can be extended by adding the gNBXe modules into the system bus architecture.

Ramirez-Agundis *et al*. [13] proposed a modular, massive-parallel, SOM-based vector quantizer for real-time video coding. The hardware architecture is divided into three sections: the processing units array, the address generator, and the control unit. The processing units array is distributed in modules, with 16 units each and a maximum of 16 modules (up to 256 neurons). Input vectors are stored in an external memory. To process one input vector, its elements are read from the memory and applied to the input of the network, one element at a time, until the complete vector is scanned. At the training stage, the scan is done twice. The first scan is to determine the winner neuron, whereas the second one is to update the weights. Each module consists of 16 processing units and a comparator to identify the local winner neuron inside that module. The control unit determines the global winner. The maximum frequency obtained after the place and route was 71.43 MHz when the design was implemented on an FPGA device.

E. Scalable/Flexible Architecture

A joint research group from the University of Lorraine and the University of Sousse has proposed hardware SOM architecture that uses network-on-chip (NoC) communication as an alternative communication approach [18], [19], [22], [40], [76]. The NoC is a network-based communication subsystem and is presented as an alternative to a traditional shared bus allowing the connection of several PEs on a single chip [77], [78]. The NoCs enjoy an explicit parallelism, high bandwidth, and a high degree of modularity, which makes them very suitable for distributed architectures such as SOM networks. The common structure of a 2-D mesh NoC is composed of the same number of PEs and routers. The data are transmitted among neurons by using packets and well-defined protocols and routing policy (i.e., wormhole routing and *XY* routing algorithm). The architecture can define its communication links dynamically by programming the packets. Its significant feature is that the NoC based architecture can perform different applications in a time-sharing manner by dynamically defining the use of neurons and their interconnections [76]. The proposed hardware SOM architecture also employs the systolic way of data exchange through the NoC, which provides high flexibility and scalability.

Recently, Hikawa [26], [69] proposed a hierarchical architecture, called nested architecture. In this layered architecture, the top module is made of four submodules placed in the second layer, and each submodule is made of smaller sub-submodules in the third layer. Each module in the bottom layer is made of four neuron units. Consequently, this architecture provides high expandability where, only by adding a module, the number of neurons is quadrupled.

F. Vector Representation

Researchers have been developing hardware SOMs and their components using digital or analog signals and techniques to represent the vector element values. Some HW SOM components are designed by using different analog techniques (i.e., current mode circuits [51] and WTA classifiers [79]). On the other hand, for a complete HW SOM implementation, all of the hardware implementations found in the literature so far employed digital techniques, as summarized in Table III. This implementation preference can be explained by the fact that digital implementation can take advantage of the benefits of the state-of-the-art VLSI and ULSI techniques [80] and powerful design environment tools and methods, such as the hardware description languages (HDL) and the easy accessibility of FPGAs. On the contrary, the primary disadvantage of analog implementations is low design flexibility even though it can possibly provide higher speed with lower hardware cost.

1) Analog: As mentioned before, some computing components implemented with analog techniques have been reported so far. However, to the best of our knowledge, no full analog implementation of SOM can be found in the literature. Here, the full implementation of SOM is assumed to include both the learning and the recall capabilities. Indeed, it is hard for the analog implementations to provide the learning mechanism of the SOM, where the weight values must be adjusted according to the input vectors. In addition, it is difficult to build and design an analog circuit that holds a number of analog weight values that are also programmable.

Dlogosz *et al.* [51] proposed analog components for the SOM, which includes an analog current-mode circuit for distance calculation between the weight and input vectors and a new analog programmable neighborhood mechanism, providing the triangular neighborhood function [64].

Shah *et al*. [79] presented a vector classifier based on a vector-matrix multiplier (VMM) and a winner-take-all (WTA) classifier structure. The proposed classifier was implemented on a large-scale system-on-chip (SoC) field-programmable analog array (FPAA). The implemented design is a mixed-mode system, including the analog classifier data path and the control circuitry for weight updates which is done with the microprocessor available on the SoC FPAA.

2) Digital: The most effective solution to design the hardware SOM is to implement the whole system in silicon with

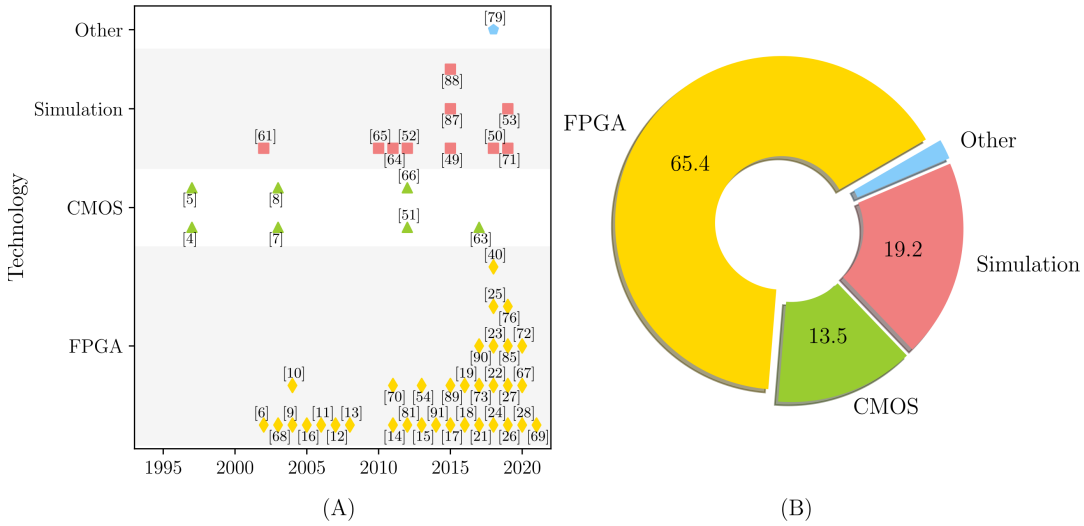

Fig. 11. Technologies in HW SOM implementations (1995–2021): (a) overview and (b) percentage chart.

digital technology. As shown in Fig. 11 and Table III, we seem to have two choices of technologies: complementary metal–oxide–semiconductor (CMOS) implementation or FPGAs.

The T0 processor for the Spert-II vector microprocessor system that implemented the SOM was fabricated with 1-μm CMOS design rules [4]. Similarly, the GENES IV processor was designed in CMOS 1-μm standard-cell technology, which is the basic block of the MANTRA I system implementing the SOM with the mantra algorithm [5]. The NBX processor was synthesized with the 0.8-μm CMOS standard-cell process and was used to build the MoNA system to implement the SOM [8]. Kim *et al.* [63] implemented their HW SOM with 65-nm CMOS technology, used for electrocardiogram (ECG) clustering.

Key design issues for the implementation technology are area, signal delay, and power consumption. The CMOS implementation is superior to the FPGA in terms of all these design issues. In contrast, big advantage of the FPGA technology is its reconfigurability and smaller nonrecurring engineering (NRE) costs. Use of the FPGA can be viewed as an intermediate solution between the ASIC and the software approach. In terms of the area, speed, and power consumption, the recent families of the major FPGA vendors have been improved significantly, which has caused a huge increase in their use in the various hardware designs. Motivated by the high speed and the low cost of parallel computation, numerous studies have employed the FPGA technology as the hardware platform to implement the SOM. As it has been shown in Fig. 11, almost two thirds of the HW SOM implementations have been implemented by using the FPGA technology. Moreover, this trend is especially valid in the last five years.

In digital design techniques, fixed-point binary representation is usually used to reduce the computing cost and memory usage, instead of floating-point representation. The crucial issue of the binary representation is the data size affecting the obtained precision. In addition, the reduction of the number of bits has an influence on the circuit size of the computing components, which significantly affects the number of neurons that can be synthesized on a single FPGA device.

Dlugosz *et al.* [81] studied the allowable reduction of the number of bits of some internal signals in an HW SOM implementation that does not deteriorate the SOM network behavior. To determine the influence of the bit lengths of signals on the quality of learning process, a series of simulations was completed using the accurate software model of the SOM implemented on an FPGA device. They revealed that the length of some internal signals can be shortened to 7 bits without disrupting the learning process and deteriorating the overall quantization error. Consequently, by shortening the data length of some internal signals, the number of neurons that can be implemented on a single device increases by 240% compared to the case where the resolution was unchanged from 16 bits.

A tristate SOM as a resource-efficient architecture for implementation on FPGA was proposed by Appiah *et al.* [52]. The tristate SOM maintains the tristate weights with {0, 1,#} as the possible values, where "#" represents "don't care" state. In addition, a modified version of the Hamming distance is used to compare input to weight vectors. Since the weight takes only three states, each state could be represented by a two-bit binary code.

Kleyko *et al.* [53] presented a generalization of the tristate self-organizing maps. In the proposed SOM, weights are allowed to be updated beyond $[-1, 1]$ to the wider range of $[-\kappa, \kappa]$. A clipping function is used as a nonlinear activation function, which is applied to all weights of each neuron. The proposed SOM achieves a better accuracy in a classification task when compared to the original tristate SOM.

3) Pulse-Based Architecture: One of the important objectives in brain modeling is to explain how the organizational order emerges by itself in the various brain maps. Kohonen [82] demonstrated that the SOM has a similar self-ordering function to that found in the biological brain. In biological neural systems, information is conveyed by electric pulses. Neural network hardware architecture that uses pulse signals

to perform the neuron's computation has been proposed and investigated [83]. A popular approach for implementing hardware neural networks is a stochastic computation [84]. The advantage of stochastic computing is that the computing elements can be realized with a smaller circuit than the conventional arithmetic ones. In stochastic computing systems, a global clock provides a time interval during which each weight value is defined. Pulse stream signal and its pulse density, i.e., probability of the presence of pulse during the interval, are used to represent a vector value. Thus, various computations can be implemented with a simple gate circuit.

Moran *et al*. [85] proposed a novel HW/SW hybrid SOM implementation using stochastic computing. Several stochastic block designs were implemented as the squared Euclidean distance and the WTA similarity check hardware. The proposed SOM was implemented on a hard processor system (HPS) and an FPGA. The WTA unit, implemented on the FPGA, carries out the WTA computation and returns the winning neuron index to the HPS. Then, software running on the HPS performs the weight update of all neurons. In this way, the SOM learning is carried out within a combined HW/SW codesign.

As another pulse-based SOM architecture, Hikawa [16] proposed a hardware SOM that uses the phase-modulated pulse signal to represent the weight value of neurons. The elements of the input and weight vectors are given as the phase of the input and internal carrier signals, and a digital phase-locked loop (DPLL) is employed as a computing element because the operation of the DPLL is very similar to that of the SOM's computation. The vector distance is computed as a phase difference of the two signals, and the winner is found by the binary-tree type BMU search circuit. Another DPLL-based SOM is proposed in [86]. It employs a new WTA method, in which the winner neuron is determined by the competition between the neurons. In all the neurons, the similarity of the carrier signals is given as pulse signals. The accumulation of these pulse signals is carried out within a digital counter. The neuron whose counter overflows first is chosen as the winner. To determine the winner, these neurons compete with each other for the time it takes for the counter to overflow. The winner neuron then spreads an update pulse signal, and its frequency is halved when it goes through other neurons. As a result, the negative-power-of-two type of neighborhood function is implemented. The proposed winner search method does not require global communication between neurons, which makes the architecture scalable.

Hikawa [87] proposed a vector classifier based on the WTA neural network (WTANN). In the WTANN, its input and weight vectors are represented by frequencies of carrier signals, and the weight update is carried out with a digital frequency-locked loop (DFLL). The winner search operation is implemented by using frequency comparators distributed among all neurons, allowing to easily increase the number of neurons in the proposed approach. In [88], a modified winner search method is proposed, where a cycle slip detector is employed to estimate the frequency difference of the signals. Since the number of cycle slips increases in proportion to the frequency difference of the two signals, the similarity between the signals in the frequency is assessed by counting the cycle slips. A VHDL simulation revealed that accuracy in the WTA operation of the proposed method is much better than the frequency comparator-based WTA circuit. However, the abovementioned WTANNs are not the classical SOM implementations since they do the learning without the neighborhood function, thus violating the topology preservation. On the other hand, the proposed WTANNs were used to test and demonstrate the use of the frequency-modulated pulse signal in the WTA function, i.e., vector distance computation and BMU search. In addition, the use of the frequency-modulated signal with the DFLL was also adopted to build SOM hardware in [25]. In the proposed SOM, the vector values are conveyed by the frequency-modulated signals and the DFLL is used for the neuron's computation. For the winner search, a cycle slip detector is also employed. In [54], this DFLL-based SOM is improved by employing the triangular neighborhood function. This function is implemented by using pulsewidth-modulated signals spread from the winner neuron without multipliers.

G. Off-Chip Learning Architecture

The works listed in Table III can be divided into two categories: the first category is called on-chip learning where the learning algorithm is also implemented in hardware; the second one, called off-chip learning, where the HW implementations belonging to this category perform only the recall operation. The recall operation, as shown in Algorithm 1, requires only the BMU search without the weight vector update operation. Thus, the vector distance calculation and the argmin function are only implemented in hardware. In the off-chip learning approach, weight vectors are trained beforehand, often by a software running on a computer, and then are loaded into the hardware to perform the recall operation.

Li *et al*. [89] developed an SOM-based positioning scheme for continuous crystal-based PET detector. The SOM training phase was accomplished off-line by MATLAB software and the test phase of the scheme was implemented on FPGA. Similarly, Hikawa *et al*. [90] proposed a hardware hand sign recognition system with a hybrid network called SOM-Hebb classifier. The weight vectors were trained by an off-chip computer, and the SOM implemented on FPGA performed the recall function only. Other examples of hardware SOMs working in the recall mode were proposed by Kurdthongmee [91] and Huang *et al*. [20] for image compression. In the former work, a memory-based BMU search unit was proposed, resulting in a reduced final number of colors. In Huang's work, a reconfigurable complete-binary-adder-tree (RCBAT), where the reuse of the employed arithmetic units is possible, was devised to reduce the hardware usage. In addition, by distributing the codebook into parallel PE blocks, the proposed design successfully demonstrated a high compression speed up to 500 frames/s.

V. Implementation and Applications

A. Implementation Platforms

To demonstrate or investigate the proposed HW SOM architectures or their computing components, authors use

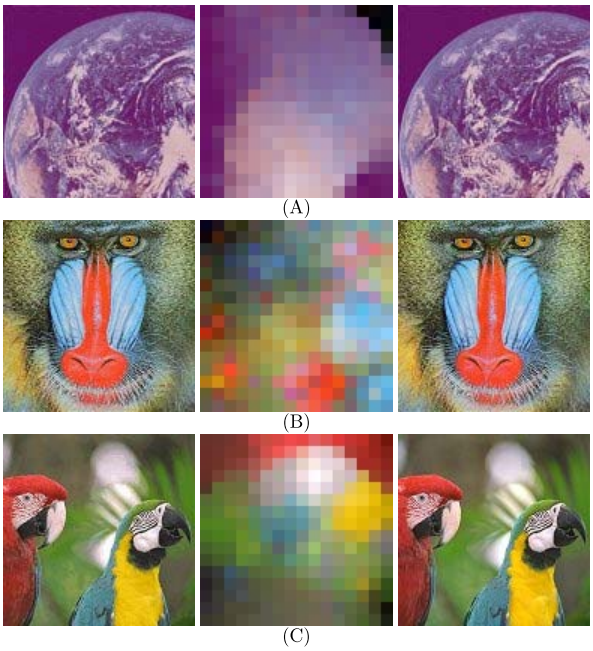

Fig. 12. Color quantization results. Original picture (left), generated color palette (middle), and reconstructed picture (right). (a) Earth, PSNR = 43.16 dB, (b) Mandrill, PSNR = 32.30 dB, and (c) Parrots, PSNR = 32.82 dB.

various methods and tools, including simulation and/or physical experimental validations. Some of the works presented in Fig. 11 and Table III were investigated by simulations (almost one fifth of the proposed HW SOM implementations). Among simulation validation tools, we can find examples of VHDL simulation [87], [88], HSpice simulation [65], and MATLAB simulation [71]. The simulation verification is very flexible and useful approach in order to validate the algorithms or circuits before their physical implementation. On the other hand, physical experimental verification is the common way of validating HW SOM implementations, where CMOS and/or FPGA technologies are targeted (almost 80% of the reported works in Fig. 11 and Table III).

B. Datasets/Target Applications

Table III summarizes the used datasets and target applications for validation of the proposed HW SOM architectures: artificial datasets, image coding and compression experiments, and classification tasks on publicly available datasets (MNIST, IRIS, and so on). From Table III, it can be noticed that most of the proposed HW SOM architectures were validated by using in-lab artificial datasets, which often represents a simple dataset to confirm the SOM's basic functions, such as the topology-preserving nature.

Many HW SOMs are applied to image coding or image compression applications. The main reason for this type of validation is that the obtained results are easily observable. Indeed, in these applications, the weight vectors are often represented as colors from the input dataset, which are the used images. In a true color digital image, each pixel is represented by R-G-B components, whose intensity is coded with 24 bits (8 bits per component). Thus, the size of a high-resolution image can be quite large (a number of pixels $\times$ 24 bits). In image compression, the main goal is to reduce the overall data size of manipulated images for storage or transmission purposes. One of the methods to compress an image is to reduce the total number of colors used to code it, which is commonly called color compression. The color compression limits the number of colors in an image and can be done with SOMs. Indeed, by presenting random input vectors (read pixels) from an image, the SOM selects the best colors from it by using its vector quantization capability. The result of this operation is a palette of the limited number of the most representative colors in the input image. This palette simply represents the SOM weights of the trained map. To produce the compressed image, each pixel of the initial image is replaced with the index of the most similar color from the SOM palette (obtained through the SOM recall operation). Thus, instead of using 24 bits to code a pixel, each pixel is coded with the number of bits necessary to uniquely identify each neuron in the map (i.e., 8 bits for a 16 $\times$ 16 map). An example of the color compression with an SOM is shown in Fig. 12 [17]. The original image, the color palette consisting of 256 colors and obtained after training a 16 $\times$ 16 SOM network, and image reconstructed by using the palette colors are shown. In addition, instead of using only one pixel as the input vector of the SOM map, $B \times B$ blocks of pixels can be presented as input vectors and quantized by the SOM in the same way as previously presented. The HW SOMs in [13], [17], [18], [20], [28], [40], [67], [68], [70], [71], [76], [91] were applied to the real-time color/image compression.

Another popular application for validation of HW SOMs is various classification tasks. In [4], a speech coding benchmark provided by EPFL was used to measure the training performance of the proposed SOM. Appiah *et al.* [52] applied their HW SOM to recognize handwritten digits. The authors used the Modified National Institute of Standards and Technology (MNIST) database [92], which is a database of handwritten digit images. In this database, each image is a 28 $\times$ 28 greyscale image, which can be considered as an input vector whose dimension is 28 $\times$ 28 = 784. An example of the MNIST data clustering [69] is shown in Fig. 13, where 784-D vectors were mapped onto a lower two-dimensional 16 $\times$ 16 SOM. It can be seen that due to the SOM topology-preserving nature, the same digit characters are assigned to adjacent neurons. Note also that clusters representing different digits but similar shape are assigned to neighboring neurons. In this way, the SOM can be used to visualize relations among input vectors. Other examples of the use of the MNIST dataset also for recognition or clustering applications can be found in [53], [69], [71]. Another example of classification tasks used to validate an HW SOM architecture is the hand sign recognition system proposed in [90]. In the proposed system, preprocessed input images were applied to the off-line trained SOM used in the recall mode allowing to achieve real-time hand sign recognition (60 frames/s with a recognition accuracy of 97.1%). Moreover, the IRIS dataset has also been used for many statistical classification techniques in machine learning algorithms and HW SOM validation as well. The IRIS dataset consists of 50 samples from each of three species of Iris

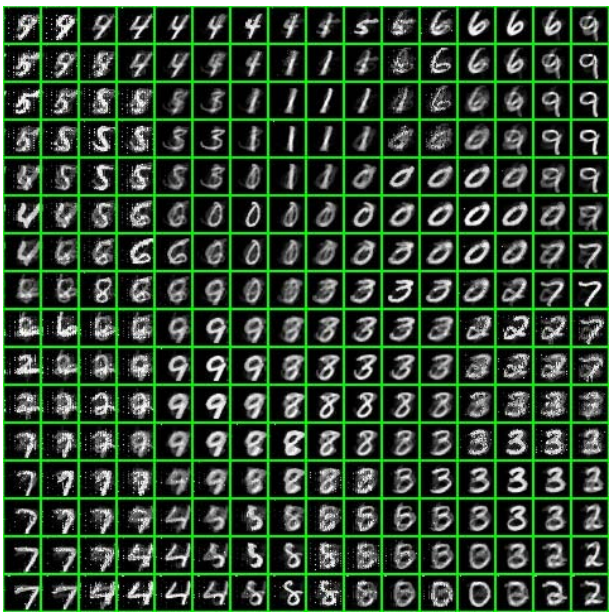

Fig. 13. MNIST clustering example.

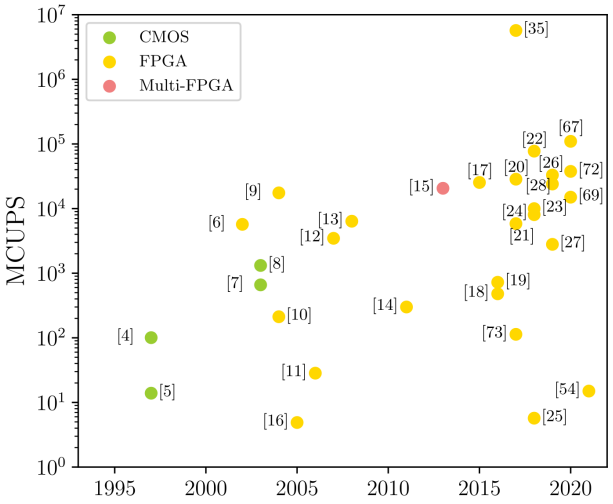

Fig. 14. Overview of performances of the HW SOM architectures in million of CUPS (MCUPS) (1995–2021).

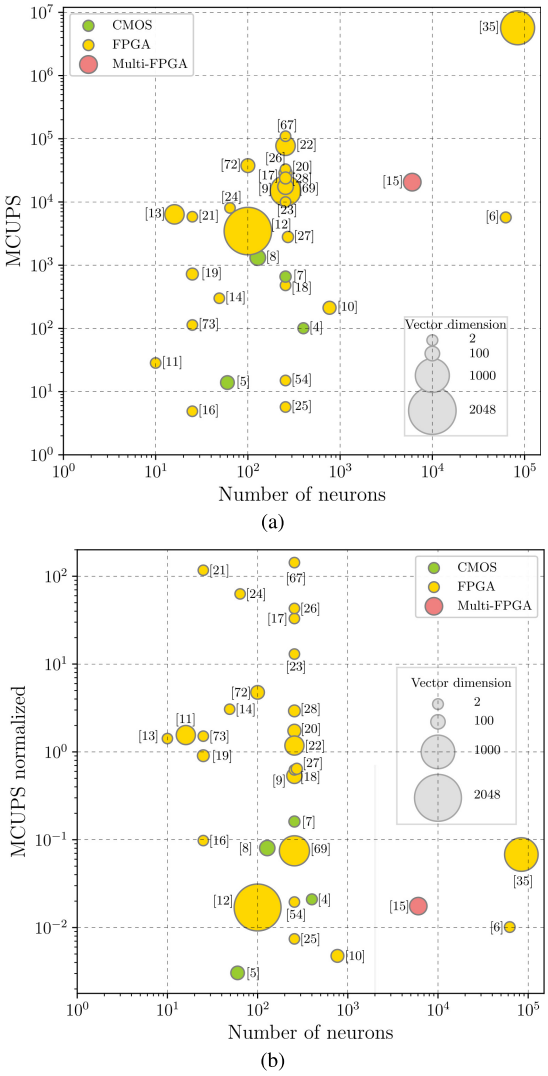

Fig. 15. Overview of performances of the HW SOM architectures in million of CUPS (MCUPS) as a function of the number of neurons and input vector dimension: (a) raw values and (b) normalized by the number of neurons and input vector dimension.

(Iris setosa, Iris virginica, and Iris versicolor). Each sample consists of four features: the length and the width of the sepals and petals, in centimeters. Classification performances of the works in [85], [87] were demonstrated by using the IRIS dataset. Finally, the mixed-mode classifier proposed by Shah *et al.* [79] was trained to identify the sound source, whether it is a generator, truck, or car.

Another data clustering application based on the SOM's vector quantization capability is presented in [15]. Lachmair *et al.* [15] used hyperspectral image data of the lunar crater volcanic field (LCVF) in Nevada, taken by NASA's AVIRIS sensor for clustering target. Since the LCVF dataset is fairly large with the high-dimensional vectors (in the hyperspectral image) and it has complex cluster structure, the clustering of these hyperspectral images is seen as a quite challenging and nontrivial task. Similarly, Kim *et al.* [63] used the proposed HW SOM implemented in 65-nm CMOS technology for electrocardiogram (ECG) clustering or de Abereu de Sousa *et al.* [24] proposed an FPGA-based SOM to detect 64-quadrature amplitude modulation (QAM) symbols, commonly applied in many communication systems. The use of SOM in the quadrature/in-phase pair detection allows the continuous adjustment to the QAM constellation with no supervision. Consequently, bandwidth savings are obtained without the need for training data retransmission.

C. Performance Measure

The processing speed of the HW SOM architectures is usually evaluated by connection updates per second (CUPS). This metric quantifies the number of weight updates that the SOM system performs per second during the learning process

$$\text{CUPS} = \frac{D \cdot N}{T_I} \cdot f \tag{11}$$

TABLE III
SOM RELATED WORKS IN THE LITERATURE

Work	Architecture	Vector distance	Neighborhood function	SOM size	Vector dimension	Precision (bit)	Technology	MCUPS	Application data set
[4]	Vector processor	Manhattan	Triangular	20×20	12	16	CMOS	100.6	EPFL speech coding benchmark
[5]	Systolic array	Manhattan	Rectangular	6×10	76	16	CMOS	13.9	Power system security assessment
[6]	Modular	Manhattan	Negative powers of two	250×250	9	16	FPGA (xcv3200e)	5700	NA
[61]	SIMD PE array	Manhattan	NA	NA	NA	NA	Theoretical study	NA	NA
[68]	SIMD PE array	Manhattan	Not used	$2 \times 2 \times 2$	3	8	FPGA (v812efg900)	NA	Color compression
[7]	SIMD array	Manhattan	Rectangular	16×16	16	8	CMOS	660	NA
[8]	Modular (SIMD PE NBX)	Manhattan	Negative powers of two	16×8	128	8	CMOS	1318	NA
[9]	Distributed	Manhattan	Negative powers of two	16×16	128	16	FPGA (xc2v6000)	17500	Image enlargement
[10]	Modular (SIMD PE/gNBX)	Manhattan	Precalculated	24×32	58	16	FPGA (xc2v4000)	212	Artificial data set
[11]	Distributed	Manhattan	Negative powers of two	10	2	8	FPGA (xc2s400)	28.38	Artificial data set
[12]	Systolic array	Manhattan	Negative powers of two	100	2048	8, 12	FPGA (xc2v6000)	3467	NA
[13]	Modular	Manhattan	Negative powers of two	256	16	8	FPGA (xc2v6000)	6372	Image compression
[14]	Distributed	Manhattan	Triangular	7×7	2	NA	FPGA (xc5vlx110t)	300	NA
[70]	Custom processor	Manhattan	Negative powers of two	16×16	3	NA	FPGA (xc2vp30ff896)	NA	Color compression
[52]	Custom processor	Modified Hamming	Rectangular	100 40	784 768	NA	Simulation xc4vlx160	NA	MNIST recognition Surveylance system Tri-state SOM
[81]	Custom processor	NA	Triangular	3×3 10×10	2	7, 10, 16	FPGA (xc5vlx110t)	NA	Artificial data set
[15]	Modular (PE gNBXe)	Squared Euclidean	Step function	6050	194	16	Five FPGAs (Virtex-4FX100s)	20604	LCVF data
[16]	Distributed	Phase error count	Precalculated	5×5	2	10	FPGA (EP20K400)	4.89	Artificial data set
[17]	Distributed	Manhattan	Negative powers of two	16×16	3	16	FPGA (xc6vsx315t)	25344	Artificial data set Color compression
[18]	NoC & distributed	Squared Euclidean	Negative powers of two	16×16	3	8	FPGA (xc7vx485T)	480	Color compression
[19]	NoC & distributed	Squared Euclidean	Negative powers of two	5×5	32	NA	FPGA (Virtex-6)	724	NA
[21]	Custom processor	Manhattan	Negative powers of two	25	2	8	FPGA (xc6vlx75t)	5847.5	Artificial data set
[73]	Distributed	Manhattan	Negative powers of two	16 25	2 3	16 16	FPGA (xc5vlx50t)	76.16 113.25	NA
[63]	Distributed	Euclidean	Rectangular	5×5	128	10	CMOS	NA	ECG clustering
[24]	Custom processor	Manhattan	Negative powers of two	64	2	16	FPGA (Kintex-U-XCKU035)	8050	QAM detection
[22]	NoC & distributed	Squared Euclidean	Negative powers of two	16×16	256	NA	FPGA (VC707 Virtex-7)	77391	NA
[23]	Disitributed	Manhattan	Negative powers of two	16×16	3	16	FPGA (xc7vx690tffg)	9984	Artificial data set
[25]	Distributed	Count of cycle slips	Programmable	16×16	3	16	FPGA (xc6vsx315t)	5.71	Artificial data set
[40]	NoC & distributed	Squared Euclidean	Negative powers of two	10×10 $\sim 7 \times 7$	$3 \sim 12$	8	FPGA (Xilinx VC707)	NA	Image compression
[26]	Distributed (Nested)	Manhattan	Negative powers of two	16×16	3	16	FPGA (xc7vx690tffg)	33024	Artificial data set
[27]	Custom processor	Inner product	Negative powers of two	272	16 (1st layer) 256 (2nd layer)	NA	FPGA (xczu9eg)	2796	Amygdala model
[85]	Distributed	Squared Euclidean	NA	9	4	8	FPGA (DE10-Nano)	NA	Iris classification
[76]	NoC & distributed	Squared Euclidean	Negative powers of two	$6 \times 6 \sim$ 15×15	$3 \sim 12$	8	FPGA (VC707 Virtex-7)	NA	Image compression
[71]	Custom processor	Manhattan	Triangular	100×100 20×20 16	3 784 16	8	MATLAB simulation	NA	Color map MNIST labelling Image compression
[53]	Algorithm	dot product cosine similarity	NA	$20 \sim 100$	784	NA	Model simulation	NA	MNIST recognition
[28]	Systolic array	Squared Euclidean	Negative powers of two	16×16	32	8	FPGA (xc7vx485t)	23997	Image compression
[67]	Custom processor	Manhattan	Not used	256	3	16	FPGA (xc7vx690)	109800	Image compression
[72]	Custom processor	Manhattan	Negative powers of two	10×10	79	16	FPGA (xcvu440)	37620	Human actions video categorization
[69]	Disitributed (Nested)	Manhattan	Negative powers of two	16×16	784	16	FPGA (xc7vx690tffg)	15012	MNIST clustering
[54]	Distributed	Count of cycle slips	Triangular	16×16	3	16	FPGA (xc6vsx315t)	15	Artificial data set
[91]	Custom processor	Manhattan	NA	16×16	NA	NA	FPGA (xc4vlx200)	NA	Color compression
[89]	Custom processor	Euclidean	NA	NA	NA	NA	FPGA (Arria II)	NA	Positioning scheme for PET detector
[90]	Custom processor	Manhattan	NA	256	32	16	FPGA (xc3s700A)	NA	Real-time hand sign recognition
[20]	SIMD PE array	Squared Euclidean	NA	256	64	8	FPGA (Stratic IV)	28494 MCPS	Image compression
[65]	Neighborhood mechanism	NA	Triangular	NA	NA	NA	Model simulation	NA	NA
[64]	Neighborhood mechanism	NA	Triangular	NA	NA	Analog	CMOS Model simulation	NA	NA
[51]	Vector dist. calc. circuit	Manhattan	NA	NA	NA	Analog	CMOS Model simulation	NA	NA
[66]	Neighboring mechanism	NA	Triangular	NA	NA	NA	CMOS (Hspice simulation)	NA	NA
[49]	Initialization of neuron weights	Euclidean Manhattan	Rectangular Triangular	$4 \times 4 \sim$ 32×32	2	NA	Software model simulation	NA	NA
[87]	WTANN (Winner search circuit)	Freq. comparison	NA	8 3 3	3 4 13	16	VHDL simulation	NA	Artificial data IRIS data WINE data
[88]	WTANN (Winner search circuit)	Cycle slip count	NA	8	3	16	VHDL simulation	NA	Artificial data
[79]	VMM + WTA	Analog vector matrix multiplication	NA	3	12	14	SoC FPAA	NA	Sound source identification (generator, truck, or car)
[50]	Weight init. circuit	Euclidean Manhattan	Rectangular Triangular	$4 \times 4 \sim$ 32×32	2	NA	Model simulation	NA	NA
[35]	Modular (SIMD PE NBX)	Squared Euclidean	Precalculated	84000	1000	16	FPGA (xcv7fx690t)	5700000	Artificial data set

where D is the input vector dimension, N is the total number of neurons in the SOM network, T_I is the total time in clock cycles needed to finish one learning iteration, and f is the maximal operating frequency of the HW architecture.

The performances of the HW SOM architectures reported in the literature so far are shown in Figs. 14 and 15 and in Table III in terms of million of CUPS (MCUPS) for a period (1995–2021); in terms of million of CUPS (MCUPS) as a function of the number of neurons and input vector dimension (raw and normalized values); and in the column "MCUPS." From the presented results, it can be noticed that the highest performance of 5.7 GCPS was achieved by the approach proposed by Lachmair *et al.* [35]. It should be mentioned that this reported value is for MCPS (evaluated only in the recall phase without updates) and not for MCUPS. Thus, to have an order of MCUPS, this value should be at least divided by a factor of 2. The second best performance of 109 800 MCUPS was achieved by the AW-SOM [67]. This is not surprising at all and is mainly due to the modified SOM algorithm, which is at the heart of this approach: no use of any neighborhood function and no BMU search at all. A small reminder is that the BMU search operation is the biggest performance bottleneck of all HW SOM architectures. Moreover, the AW-SOM does not have all the features provided by the conventional SOMs because the neighborhood function and topological relations of the neurons were also omitted. Another way of presenting the performances of the HW SOM architectures is to normalize the values of MCUPS by the number of neurons and input vector dimension. These results are presented in Fig. 15(b). If we look at 11, the normalized MCUPS presents only the ratio between the maximum operating frequency and the time needed to finish one learning operation. This value is more representative as it shows the most optimized HW SOM architectures (the higher is better).

In addition, it should also be noted (see Fig. 14) that the performances of the most recent HW SOM implementations are almost all in a range between 10 and 100 Giga CUPS. This can be explained by the use of the most recent FPGA devices and in the adoption of the architectural choices discussed earlier in this article. It should also be highlighted that the performance of the hardware SOM with the off-chip architecture presented in [20] was given in Figs. 14 and 15 and Table III in connections per second (CPS) (evaluated only in recall phase).

VI. Open Research Problems

From the previous discussion on the different aspects of the HW SOM architectures found in the literature, it can be concluded that significant improvements have been achieved in the last few decades at all, circuit, algorithmic, and architectural levels. Consequently, these improvements pave the way to use the SOM algorithm in a large variety of applications with specific data and computing requirements [93]–[98].

Although the SOM algorithm is not new, its simplicity and capability to tackle a large variety of problems starting from vector quantization and clustering through data visualization and image/video processing keep it attractive now more than ever, especially nowadays where the big data (BD) and Internet of Things (IoT) infrastructures are at the origin of a tremendous amount of data of different types, which is continuously generated and supplied by a myriad of sensors deployed everywhere.

The possibility to make HW SOMs ubiquitous and easily accessible as HW IPs and accelerators in many application fields (and HW platforms) goes through the solution of some still open research and challenging problems, which can be categorized into four groups.

1) *Huge Scalable HW SOM:* From the recent state-of-the-art works, it has been shown that the new trends in the design of HW SOMs are oriented toward scalable and expandable architectures [18], [19], [22], [26], [40], [69], [76]. This current trend is in line with the explosive growth of data volumes we are witnessing these last few years where very large SOM architectures in terms of neurons are necessary to satisfy these huge volume data-related needs. The design of new high-performance scalable and expandable HW SOM architectures will allow to tackle this problem easier where these new architectures will be used as the basic building blocks for larger networks of different size.
2) *Fast BMU Search:* High degree of scalability and the possibility to build huge HW SOMs put in the forefront the problem of BMU search in such networks, the operation which is even in today's HW SOM architecture identified as the biggest performance bottleneck. Potential alternatives could be newly adapted SOM algorithms as the one proposed in AW-SOM [67] or even to shorten the BMU search operation by exploiting the inherent topology preservation property of SOMs [99].
3) *High Configurability:* The new HW SOM architectures have to be not only as much as possible high performance and scalable but also highly configurable and adaptable to a large variety of applications. Ideally, the new HW SOM architectures should be application-agnostic and all application-specific needs should be provided as a simple list of online and real-time configurable parameters.
4) *Toward Growing HW SOM-Models:* High configurability of the HW SOM architectures implies also to have the possibility to change the number of neurons dynamically or to adapt it to learning requirements of a given application. Consequently, dynamic and growing counterparts of the SOM algorithm should be targeted in HW SOM architectures. For instance, the pioneering work presented in [100] where a growing grid (GG)-based algorithm has been implemented in hardware is an example of these dynamic and growing HW architectures. Ideally, the growing neural gas (GNG)-based models that are quite challenging and more advantageous over other growing models should find their implementations in HW in the near future.

VII. Conclusion

In this article, a survey of hardware SOM implementations has been presented. The SOM algorithm, whose simplicity and capability to tackle a large variety of problems (vector quantization, clustering, data visualization, image/video processing, and so on), is still attractive in many application fields and is considerably gaining more attention in nowadays's BD and IoT infrastructures. Its inherent property of topology preservation

and unsupervised learning of processed data without any prior knowledge put it in the front of candidates for data reduction. However, its high computational cost makes the online real-time high-performance SOM processing mostly reserved for specific hardware implementations. In this article, we gave an overview of the hardware, application-specific implementations of the SOM algorithm, the most widely used computing blocks, architectures, design choices, adaptation, and optimization techniques that have been reported in the literature so far in the field of hardware SOMs. Moreover, an overview of main challenges and trends for their ubiquitous adoption as hardware accelerators in many application fields has also been addressed. This article is expected to be useful for researchers in the area of artificial intelligence in a broader sense, real-time hardware architecture, and system design with tight design constraints in terms of performance (timing), power, and area.

REFERENCES

[1] T. Kohonen, *Self-Organizing Maps*. New York, NY, USA: Springer-Verlag, 2001.

[2] U. Asan and S. Ercan, "An introduction to self-organizing maps," in *Computational Intelligence Systems in Industrial Engineering*, vol. 6, Cengiz Kahraman, Ed. Paris, France: Atlantis Press, 2012, pp. 295–315.

[3] G. K. Matsopoulos, *Self-Organizing Maps*. Vukovar, Croatie: InTech, 2010.

[4] K. Asanovic, "A fast Kohonen net implementation for spert-II," in *Proc. Int. Work-Conf. Artif. Natural Neural Netw. (IWANN)*, 1997, pp. 792–800.

[5] P. Ienne, P. Thiran, and N. Vassilas, "Modified self-organizing feature map algorithms for efficient digital hardware implementation," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 315–330, Mar. 1997, doi: 10.1109/72.557669.

[6] M. Porrmann, M. Franzmeier, H. Kalte, U. Witkowski, and U. Rückert, "A reconfigurable SOM hardware accelerator," in *Proc. Eur. Symp. Artif. Neural Netw. (ESANN)*, Apr. 2002, pp. 337–342.

[7] D. C. Hendry, A. A. Duncan, and N. Lightowler, "IP core implementation of a self-organizing neural network," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1085–1096, Sep. 2003, doi: 10.1109/TNN.2003.816353.

[8] M. Porrmann, U. Witkowski, and U. Ruckert, "A massively parallel architecture for self-organizing feature maps," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1110–1121, Sep. 2003, doi: 10.1109/TNN.2003.816368.

[9] H. Tamukoh, T. Aso, K. Horio, and T. Yamakawa, "Self-organizing map hardware accelerator system and its application to realtime image enlargement," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 4, Sep. 2003, pp. 2683–2687, doi: 10.1109/IJCNN.2004.1381073.

[10] C. Pohl, M. Franzmeier, M. Porrmann, and U. Ruckert, "gNBX–reconfigurable hardware acceleration of self-organizing maps," in *Proc. IEEE Int. Conf. Field- Program. Technol.*, Dec. 2004, pp. 97–104, doi: 10.1109/FPT.2004.1393256.

[11] J. Pena, M. Vanegas, and A. Valencia, "Digital hardware architectures of Kohonen's self organizing feature maps with exponential neighboring function," in *Proc. IEEE Int. Conf. Reconfigurable Comput. FPGA's (ReConFig)*, Sep. 2006, pp. 1–8, doi: 10.1109/RECONF.2006.307761.

[12] I. Manolakos and E. Logaras, "High throughput systolic SOM IP core for FPGAs," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2007, pp. II-61–II-64, doi: 10.1109/ICASSP.2007.366172.

[13] A. Ramirez-Agundis, R. Gadea-Girones, and R. Colom-Palero, "A hardware design of a massive-parallel, modular NN-based vector quantizer for real-time video coding," *Microprocessors Microsyst.*, vol. 32, no. 1, pp. 33–44, Feb. 2008, doi: 10.1016/j.micpro.2007.06.004.

[14] R. Dlugosz, M. Kolasa, and M. Szulc, "An FPGA implementation of the asynchronous programmable neighborhood mechanism for WTM self-organizing map," in *Proc. 18th Int. Conf. Mixed Design Integr. Circuits Syst. (MIXDES)*, Jun. 2011, pp. 258–263.

[15] J. Lachmair, E. Merényi, M. Porrmann, and U. Rückert, "A reconfigurable neuroprocessor for self-organizing feature maps," *Neurocomputing*, vol. 112, pp. 189–199, Jul. 2013, doi: 10.1016/j.neucom.2012.11.045.

[16] H. Hikawa, "FPGA implementation of self organizing map with digital phase locked loops," *Neural Netw.*, vol. 18, nos. 5–6, pp. 514–522, 2005, doi: 10.1016/j.neunet.2005.06.012.

[17] H. Hikawa and Y. Maeda, "Improved learning performance of hardware self-organizing map using a novel neighborhood function," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2861–2873, Nov. 2015.

[18] M. Abadi, S. Jovanovic, K. B. Khalifa, S. Weber, and M. H. Bedoui, "A hardware configurable self-organizing map for real-time color quantization," in *Proc. IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Monte Carlo, Monaco, Dec. 2016, pp. 336–339.

[19] M. Abadi, S. Jovanovic, K. B. Khalifa, S. Weber, and M. H. Bedoui, "A scalable flexible SOM NoC-based hardware architecture," in *Advances in Self-Organizing Maps and Learning Vector Quantization*, vol. 428, E. Merényi, M. J. Mendenhall, and P. O'Driscoll, Eds. Cham, Switzerland: Springer, 2016, pp. 165–175, doi: 10.1007/978-3-319-28518-4_14.

[20] Z. Huang *et al.*, "A hardware-efficient vector quantizer based on self-organizing map for high-speed image compression," *Appl. Sci.*, vol. 7, no. 11, p. 1106, Oct. 2017, doi: 10.3390/app7111106.

[21] M. A. A. de Sousa and E. Del-Moral-Hernandez, "Comparison of three FPGA architectures for embedded multidimensional categorization through Kohonen's delf-organizing maps," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Baltimore, MD, USA, May 2017, pp. 1–4.

[22] S. Jovanovic, H. Rabah, and S. Weber, "High performance scalable hardware SOM architecture for real-time vector quantization," in *Proc. IEEE Int. Conf. Image Process., Appl. Syst. (IPAS)*, Sophia Antipolis, France, Dec. 2018, pp. 256–261, doi: 10.1109/IPAS.2018.8708863.

[23] H. Hikawa, H. Ito, and Y. Maeda, "A new hardware self-organizing map architecture with high expandability," in *Proc. IEEE Int. Conf. Image Process., Appl. Syst. (IPAS)*, Sophia Antipolis, France, Dec. 2018, pp. 238–243.

[24] M. A. de Abreu de Sousa, R. Pires, S. D. S. Perseghini, and E. Del-Moral-Hernandez, "An FPGA-based SOM circuit architecture for online learning of 64-QAM data streams," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8, doi: 10.1109/IJCNN.2018.8489518.

[25] H. Hikawa, H. Ito, and Y. Meda, "Hardware self-organizing map based on frequency-modulated signal and digital frequency-locked loop," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1–5, doi: 10.1109/ISCAS.2018.8351364.

[26] H. Hikawa, "Nested hardware architecture for self-organizing map," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Budapest, Hungary, Jul. 2019, pp. 1–7.

[27] Y. Tanaka and H. Tamukoh, "Hardware implementation of brain-inspired amygdala model," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Sapporo, Japan, May 2019, pp. 1–5.

[28] K. Ben Khalifa, A. G. Blaiech, and M. H. Bedoui, "A novel hardware systolic architecture of a self-organizing map neural network," *Comput. Intell. Neurosci.*, vol. 2019, pp. 1–14, Apr. 2019.

[29] L. Khacef, V. Gripon, and B. Miramond, "GPU-based self-organizing maps for post-labeled few-shot unsupervised learning," 2020, *arXiv:2009.03665*.

[30] Y. Xiao, C. S. Leung, T.-Y. Ho, and P.-M. Lam, "A GPU implementation for LBG and SOM training," *Neural Comput. Appl.*, vol. 20, no. 7, pp. 1035–1042, Oct. 2011.

[31] V. Mallet, M. Nilges, and G. Bouvier, "Quicksom: Self-organizing maps on GPUs for clustering of molecular dynamics trajectories," *Bioinformatics*, vol. 37, no. 14, pp. 2064–2065, Aug. 2021.

[32] P. Wittek, S. C. Gao, I. S. Lim, and L. Zhao, "Somoclu: An efficient parallel library for self-organizing maps," *J. Stat. Softw.*, vol. 78, no. 9, pp. 1–21, 2017.

[33] R. Mancini, A. Ritacco, G. Lanciano, and T. Cucinotta, "XPySom: High-performance self-organizing maps," in *Proc. IEEE 32nd Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Porto, Portugal, Sep. 2020, pp. 209–216.

[34] Y. Liu, J. Sun, Q. Yao, S. Wang, K. Zheng, and Y. Liu, "A scalable heterogeneous parallel SOM based on MPI/CUDA," in *Proc. Asian Conf. Mach. Learn.*, 2018, pp. 264–279.

[35] J. Lachmair, T. Mieth, R. Griessl, J. Hagemeyer, and M. Porrmann, "From CPU to FPGA—Acceleration of self-organizing maps for data mining," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Anchorage, AK, USA, May 2017, pp. 4299–4308.

[36] G. Bouvier, N. Desdouits, M. Ferber, A. Blondel, and M. Nilges, "An automatic tool to analyze and cluster macromolecular conformations based on self-organizing maps," *Bioinformatics*, vol. 31, no. 9, pp. 1490–1492, May 2015.

[37] V. Moosavi, S. Packmann, and I. E. S. A. Vall. (2014). *Sompy Python Library for Self Organizing Map (SOM)*. GitHub. [Online]. Available: https://github.com/sevamoo/SOMPY

[38] L.-P. Chen, Y.-G. Liu, Z.-X. Huang, and Y.-T. Shi, "An improved SOM algorithm and its application to color feature extraction," *Neural Comput. Appl.*, vol. 24, nos. 7–8, pp. 1759–1770, Jun. 2014, doi: 10.1007/s00521-013-1416-9.

[39] A. De, Y. Zhang, and C. Guo, "A parallel adaptive segmentation method based on SOM and GPU with application to MRI image processing," *Neurocomputing*, vol. 198, pp. 180–189, Jul. 2016.

[40] M. Abadi, S. Jovanovic, K. B. Khalifa, S. Weber, and M. H. Bedoui, "A scalable and adaptable hardware NoC-based self organizing map," *Microprocessors Microsyst.*, vol. 57, pp. 1–14, Mar. 2018, doi: 10.1016/j.micpro.2017.12.007.

[41] M. Porrmann, U. Witkowski, and U. Rückert, *Implementation of Self-Organizing Feature Maps in Reconfigurable Hardware*. Boston, MA, USA: Springer, 2006, pp. 247–269.

[42] D. Timo Hämäläinen, "Parallel implementations of self-organizing maps," in *Self-Organizing Neural Networks*, vol. 78, J. Kacprzyk, U. Seiffert, and L. C. Jain, Eds. Berlin, Germany: Physica-Verlag HD, 2002, pp. 245–278.

[43] T. Kohonen and P. Somervuo, "How to make large self-organizing maps for nonvectorial data," *Neural Netw.*, vol. 15, nos. 8–9, pp. 945–952, 2002.

[44] J. Vesanto and E. Alhoniemi, "Clustering of the self-organizing map," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 586–600, May 2000.

[45] M. Attik, L. Bougrain, and F. Alexandre, "Self-organizing map initialization," in *Proc. Int. Conf. Artif. Neural Netw.* Berlin, Germany: Springer, 2005, pp. 357–362.

[46] A. Fiannaca, R. Rizzo, A. Urso, and S. Gaglio, "A new SOM initialization algorithm for nonvectorial data," in *Proc. Int. Conf. Knowl.-Based Intell. Inf. Eng. Syst.* Berlin, Germany: Springer, 2008, pp. 41–48.

[47] I. Valova, G. Georgiev, N. Gueorguieva, and J. Olson, "Initialization issues in self-organizing maps," *Procedia Comput. Sci.*, vol. 20, pp. 52–57, 2013.

[48] A. A. Akinduko, E. M. Mirkes, and A. N. Gorban, "SOM: Stochastic initialization versus principal components," *Inf. Sci.*, pp. 364–365, pp. 213–221, Oct. 2016.

[49] M. Kolasa, R. Dlugosz, and W. Pedrycz, "Problem of efficient initialization of large self-organizing maps implemented in the CMOS technology," in *Proc. IEEE 2nd Int. Conf. Cybern. (CYBCONF)*, Jun. 2015, pp. 36–41.

[50] M. Kolasa, R. Długosz, T. Talaśka, and W. Pedrycz, "Efficient methods of initializing neuron weights in self-organizing networks implemented in hardware," *Appl. Math. Comput.*, vol. 319, pp. 31–47, Feb. 2018, doi: 10.1016/j.amc.2017.01.043.

[51] R. Dlugosz, T. Talaska, W. Pedrycz, and P.-A. Farine, "Low-power Manhattan distance calculation circuit for self-organizing neural networks implemented in the CMOS technology," in *Proc. Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn. (ESANN)*, Bruges, Belgium, Apr. 2012, pp. 615–620.

[52] K. Appiah, A. Hunter, P. Dickinson, and H. Meng, "Implementation and applications of tri-state self-organizing maps on FPGA," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 8, pp. 1150–1160, Aug. 2012, doi: 10.1109/TCSVT.2012.2197077.

[53] D. Kleyko, E. Osipov, D. D. Silva, U. Wiklund, and D. Alahakoon, "Integer self-organizing maps for digital hardware," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Budapest, Hungary, Jul. 2019, pp. 1–8.

[54] H. Hikawa, "Hardware self-organizing map based on digital frequency-locked loop and triangular neighborhood function," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 3, pp. 1245–1258, Mar. 2021, doi: 10.1109/TCSI.2020.3046795.

[55] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, Apr. 1987, doi: 10.1109/MASSP.1987.1165576.

[56] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, "Winner-take-all networks of O(N) complexity," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Jan. 1988, pp. 703–711.

[57] M. Oster, R. Douglas, and S.-C. Liu, "Computation with spikes in a winner-take-all network," *Neural Comput.*, vol. 21, no. 9, pp. 2437–2465, Sep. 2009, doi: 10.1162/neco.2009.07-08-829.

[58] B. Ruf and M. Schmitt, "Self-organization of spiking neurons using action potential timing," *IEEE Trans. Neural Netw.*, vol. 9, no. 3, pp. 575–578, May 1998, doi: 10.1109/72.668899.

[59] D. T. Pham, M. S. Packianather, and E. Y. A. Charles, "A self-organising spiking neural network trained using delay adaptation," in *Proc. IEEE Int. Symp. Ind. Electron.*, Jun. 2007, pp. 3441–3446, doi: 10.1109/ISIE.2007.4375170.

[60] A. Gupta and L. N. Long, "Hebbian learning with winner take all for spiking neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 2009, pp. 1054–1060, doi: 10.1109/IJCNN.2009.5178751.

[61] B. Mailachalam and T. Srikanthan, "Area–time issues in the VLSI implementation of self organizing map neural networks," *Microprocessors Microsyst.*, vol. 26, pp. 399–406, Dec. 2002, doi: 10.1016/S0141-9331(02)00065-0.

[62] W. Kurdthongmee, "A low latency minimum distance searching unit of the SOM based hardware quantizer," *Microprocessors Microsyst.*, vol. 39, no. 2, pp. 135–143, Mar. 2015.

[63] J. Kim and P. Mazumder, "Energy-efficient hardware architecture of self-organizing map for ECG clustering in 65-nm CMOS," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 9, pp. 1097–1101, Sep. 2017.

[64] R. Dlugosz, M. Kolasa, W. Pedrycz, and M. Szulc, "Parallel programmable asynchronous neighborhood mechanism for Kohonen SOM implemented in CMOS technology," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 2091–2104, Dec. 2011.

[65] R. Długosz, M. Kolasa, and K. Bielinski, "Programmable triangular neighborhood function for Kohonen self-organizing map implemented on chip," in *Proc. 17th Int. Conf. Mixed Design Integr. Circuits Syst. (MIXDES)*, Jun. 2010, pp. 328–332.

[66] M. Kolasa, R. Długosz, W. Pedrycz, and M. Szulc, "A programmable triangular neighborhood function for a Kohonen self-organizing map implemented on chip," *Neural Netw.*, vol. 25, pp. 146–160, Jan. 2012, doi: 10.1016/j.neunet.2011.09.002.

[67] G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, M. Re, and S. Spanó, "AW-SOM, an algorithm for high-speed learning in hardware self-organizing maps," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 2, pp. 380–384, Feb. 2020.

[68] N. Sudha, T. Srikanthan, and B. Mailachalam, "A VLSI architecture for 3-D self-organizing map based color quantization and its FPGA implementation," *J. Syst. Archit.*, vol. 48, nos. 11–12, pp. 337–352, Apr. 2003.

[69] H. Hikawa, "Nested pipeline hardware self-organizing map for high dimensional vectors," in *Proc. 27th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Glasgow, U.K., Nov. 2020, pp. 1–4, doi: 10.1109/ICECS49266.2020.9294973.

[70] W. Kurdthongmee, "Utilization of a rational-based representation to improve the image quality of a hardware-based K-SOM quantizer," *J. Real-Time Image Process.*, vol. 6, no. 3, pp. 199–211, Sep. 2011.

[71] Y.-H. Sun and T.-D. Chiueh, "A flexible and high-performance self-organizing feature map training acceleration circuit and its applications," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Mar. 2019, pp. 92–96, doi: 10.1109/AICAS.2019.8771556.

[72] M. A. D. A. D. Sousa, R. Pires, and E. Del-Moral-Hernandez, "SOM-processor: A high throughput FPGA-based architecture for implementing self-organizing maps and its application to video processing," *Neural Netw.*, vol. 125, pp. 349–362, May 2020.

[73] M. A. A. de Sousa and E. Del-Moral-Hernandez, "An FPGA distributed implementation model for embedded SOM with on-line learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Anchorage, AK, USA, May 2017, pp. 3930–3937.

[74] L. Rodriguez, L. Khacef, and B. Miramond, "A distributed cellular approach of large scale SOM models for hardware implementation," in *Proc. IEEE Int. Conf. Image Process., Appl. Syst. (IPAS)*, Sophia Antipolis, France, Dec. 2018, pp. 250–255.

[75] S. Y. Kung, *VLSI Array Processors*. Springer-Verlag, New York, 1998.

[76] M. Abadi, S. Jovanovic, K. B. Khalifa, S. Weber, and M. H. Bedoui, "A multi-application, scalable and adaptable hardware SOM architecture," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Budapest, Hungary, Jul. 2019, pp. 1–8.

[77] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[78] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Design Autom. Conf.*, Jun. 2001, pp. 684–689.

[79] S. Shah and J. Hasler, "SoC FPAA hardware implementation of a VMM+WTA embedded learning classifier," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 1, pp. 28–37, Mar. 2018.

[80] S. Y. Kung, *Digital Neural Networks*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
[81] R. Dlugosz, M. Kolasa, M. Szulc, W. Pedrycz, and P.-A. Farine, "Implementation issues of Kohonen self-organizing map realized on FPGA," in *Proc. Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn. (ESANN)*, Bruges, Belgium, Apr. 2012, pp. 633–638.
[82] T. Kohonen, "The self-organizing map, a possible model of brain maps," *Med. Biol. Eng. Comput.*, vol. 34, pp. 5–8, Mar. 1996.
[83] L. M. Reyneri, "A performance analysis of pulse stream neural and fuzzy computing systems," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 10, pp. 642–660, Oct. 1995, doi: 10.1109/82.471393.
[84] Y. Liu, S. Liu, Y. Wang, F. Lombardi, and J. Han, "A survey of stochastic computing neural networks for machine learning applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 7, pp. 2809–2824, Jul. 2021, doi: 10.1109/TNNLS.2020.3009047.
[85] A. Moran, J. L. Rossello, M. Roca, E. Isern, V. Martinez-Moll, and V. Canals, "Self-organizing maps hybrid implementation based on stochastic computing," in *Proc. XXXIV Conf. Design Circuits Integr. Syst. (DCIS)*, Bilbao, Spain, Nov. 2019, pp. 1–6.
[86] H. Hikawa, "DPLL based hardware SOM with a new winner-take-all circuit," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Aug. 2013, pp. 1–8, doi: 10.1109/IJCNN.2013.6707075.
[87] H. Hikawa, "Vector classification by a winner-take-all neural network with digital frequency-locked loop," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8, doi: 10.1109/IJCNN.2015.7280581.
[88] H. Hikawa, "Improved winner-take-all circuit for neural network based on frequency-modulated signals," in *Proc. IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2016, pp. 85–88, doi: 10.1109/ICECS.2016.7841138.
[89] D. Li, Y. Wang, L. Wang, and X. Cheng, "Implementation of self-organizing map based positioning scheme on FPGA," in *Proc. IEEE Nucl. Sci. Symp. Med. Imag. Conf. (NSS/MIC)*, Nov. 2014, pp. 1–3.
[90] H. Hikawa and K. Kaida, "Novel FPGA implementation of hand sign recognition system with SOM–Hebb classifier," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 1, pp. 153–166, Jan. 2015, doi: 10.1109/TCSVT.2014.2335831.
[91] W. Kurdthongmee, "A hardware centric algorithm for the best matching unit searching stage of the SOM-based quantizer and its FPGA implementation," *J. Real-Time Image Process.*, vol. 12, no. 1, pp. 71–80, Dec. 2013.
[92] *THE MNIST DATABASE of Handwritten Digits*. Accessed: Oct. 15, 2021. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[93] A. Ortiz, J. M. Gorriz, J. Ramirez, and D. Salas-Gonzalez, "Improving MR brain image segmentation using self-organising maps and entropy-gradient clustering," *Inf. Sci.*, vol. 262, pp. 117–136, Mar. 2014.
[94] L. E. Brito da Silva and D. C. Wunsch, "An information-theoretic-cluster visualization for self-organizing maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2595–2613, Jun. 2018.
[95] Y. Zhang, Y. Tang, B. Fang, and Z. Shang, "Real-time object tracking in video pictures based on self-organizing map and image segmentation," in *Proc. IEEE 7th Joint Int. Inf. Technol. Artif. Intell. Conf.*, Dec. 2014, pp. 559–563.
[96] P. Gunawardena *et al.*, "Real-time automated video highlight generation with dual-stream hierarchical growing self-organizing maps," *J. Real-Time Image Process.*, vol. 18, no. 5, pp. 1457–1475, Mar. 2020.
[97] S. Guo, J. Wang, Z. Chen, Y. Li, and Z. Lu, "Securing IoT space via hardware trojan detection," *IEEE Internet Things J.*, vol. 7, no. 11, pp. 11115–11122, Nov. 2020.
[98] S. Aly and S. Almotairi, "Deep convolutional self-organizing map network for robust handwritten digit recognition," *IEEE Access*, vol. 8, pp. 107035–107045, 2020.
[99] Y. Bernard, N. Hueber, and B. Girau, "A fast algorithm to find best matching units in self-organizing maps," in *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2020, pp. 825–837.
[100] S. Jovanovic, H. Rabah, S. Weber, K. B. Khalifa, and M. H. Bedoui, "Scalable, dynamic and growing hardware self-organizing architecture for real-time vector quantization," in *Proc. 27th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2020, pp. 1–4.

Slaviša Jovanović (Member, IEEE) received the B.S. degree in electrical engineering from the University of Belgrade, Belgrade, Serbia, in 2004, and the M.S. and Ph.D. degrees in electrical engineering from the University of Lorraine, Lorraine, France, in 2006 and 2009, respectively.

From 2009 to 2012, he was with the Diagnosis and Interventional Adaptive Imaging laboratory (IADI), Nancy, France, as a Research Engineer working on MRI-compatible sensing embedded systems. Then, he joined the Faculty of Sciences and Technologies and the Jean Lamour Institute (UMR 7198), University of Lorraine, Nancy, France, where he is currently an Associate Professor. His main research interests include energy-harvesting circuits, neuromorphic architectures, reconfigurable network-on-chips, and algorithm-architecture matching for real-time signal processing. He is the author and coauthor of more than 50 articles in conference proceedings and international peer-reviewed journals. He holds one patent.

Hiroomi Hikawa (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in electrical engineering from Keio University, Tokyo, Japan, in 1984, 1986, and 1989, respectively.

He then served as a Post-Doctoral Researcher with the University of South Florida, Tampa, FL, USA. From 1992 to 2008, he worked with the Computer Science and Intelligent Systems Department, Oita University, Oita, Japan. In 2008, he joined the Department of Electrical and Electronic Engineering, Kansai University, where he is currently a Professor. His research interests include architecture for signal processing and neural networks.