A Scalable Flexible SOM NoC-Based Hardware Architecture

Mehdi Abadi, Slavisa Jovanovic, Khaled Ben Khalifa, Serge Weber and Mohamed Hédi Bedoui

Abstract In this paper, a parallel hardware implementation of a self-organizing map (SOM) is presented. Practical scalability and flexibility are the main architecture features which are obtained by using a Network-on-chip (NoC) approach for communication between neurons. The presented hardware architecture allows online learning and can be easily adapted for a large variety of applications without a considerable design effort. A hardware 5 × 5 SOM was validated through the FPGA implementation and its performances at a working frequency of 200 MHz for a 32-element input vector reach 724 MCUPS in the learning and 1168 MCPS in the recall phase.

1 Introduction

Since their introduction, Self-Organizing Maps (SOMs) have been largely used in many applications [1]. A SOM is an unsupervised learning neural network which is mainly used to reduce and classify high-dimensional input data sets to ease their interpretation and processing. A SOM can be implemented either in software (SW), hardware (HW) or mixed hardware-software platforms (HW/SW). Even though the

M. Abadi (✉) · S. Jovanovic (✉) · S. Weber
UMR 7198, Institut Jean Lamour, Université de Lorraine, Nancy, France
e-mail: mehdi.abadi@univ-lorraine.fr

S. Jovanovic
e-mail: slavisa.jovanovic@univ-lorraine.fr

S. Weber
e-mail: serge.weber@univ-lorraine.fr

M. Abadi · K. Ben Khalifa (✉) · M.H. Bedoui
LR12ES06, Laboratoire de Technologie Et Imagerie Médicale,
Université de Monastir, Monastir, Tunisia
e-mail: khaled.benkhalifa@issatso.rnu.tn

M. Abadi
Ecole Nationale d'Ingénieurs de Sousse, Université de Sousse,
Sousse, Tunisia

E. Merényi et al. (eds.), *Advances in Self-Organizing Maps and Learning Vector Quantization*, Advances in Intelligent Systems and Computing 428,
DOI 10.1007/978-3-319-28518-4_14

software solutions provide more flexibility, the hardware implementations exploit inherent parallelism of SOM networks and may be preferred to the software ones especially in real-time applications characterized with tight temporal constraints.

The hardware SOM implementations are typically application specific. Each application has its own specificities needing different SOM parameters: input layer size, number of neurons in the output layer, timing constrains, memory requirements, etc. In most of cases, the adaptation of a hardware SOM architecture for an other application is time consuming and needs considerable design efforts. Another important issue which make difficult the reuse of an existing hardware SOM implementation are the communication links between neurons. Generally, these communication links are established at the point-to-point basis and are hard wired and if we want to add some additional neurons (and thus new connections to them) we have to completely modify the way neurones are connected. To have a scalable and application-independent hardware SOM implementation, it is necessary to add more flexibility to the existing HW design approaches. A way of doing this is to completely decouple computation from communication. In this paper, we propose a Network-on-a-chip (NoC) based solution, where a NoC is used for communication purposes between neurons.

This paper is organized as follows: Sect. 2 presents the state of the art in the domain of hardware SOM implementations. Section 3 presents the proposed method and describes the modifications that should be made to an existing hardware SOM implementation to make it scalable. Section 4 presents some obtained results whereas some conclusions and perspectives are drawn in Sect. 5.

2 Related Work

The first reported SOM implementations were in software using processor-based architectures [2]. The performances of initial single-core microprocessor architectures have been recently boosted by the increasing parallelism of many-cores multiprocessor chips (MPSoC), but are still suffering from sequential processing and high power consumption with respect to the application-specific solutions. However, the SOM software implementations are flexible and easy to implement and are usually used beforehand a hardware implementation especially in the design exploration phase to give rapidly insights about the HW design choices to take. However, for hard real-time embedded applications, hardware solutions based on the use of Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs) may be preferred.

The FPGA solutions are a good trade-off between cost, design effort, performances and reduced time-to-market. However, the FPGAs can only be used to implement digital counterparts of SOMs, no analog design is supported. If the high performances, low power consumption or low area occupancy are targeted, an ASIC is preferable to an FPGA implementation. There are some ASIC implementations of SOMs that we found in literature [3–5]. An ASIC implementation gives the best performances but is costly, demands high design efforts and has little or no flexibility.

In a hardware design, each choice has a cost and must be carefully considered. The floating point operators are resources greedy and are often avoided in hardware implementations, unless there are no solutions to obtain needed precision. Besides the arithmetic precision, in HW SOM implementations the choice of norm for distance calculation can influence the overall complexity of the hardware. Therefore, we found some architectures using Manhattan or Euclidean distance [6–8]. The first one is preferred to the latter due to the lower computation requirements and thus lower power consumption. Some studies showed that for high-dimensional vectors the effect of choosing the L1 operator is negligible on the SOM performances [9].

Another important choice to take in HW SOM implementations is the type of neighbourhood function (NF) to use, whose function determines which neurons' coefficients in the vicinity of the winning one should be updated. In the original SOM algorithm, a Gaussian neighborhood function is used, but its hardware implementation demands complex arithmetic operations and is usually realized as an analog integrated circuit [4]. It is often approximated with other functions such as: rectangular, triangular, shift-register based [5, 7]. The shift-register solution of the NF greatly simplifies its implementation by replacing the resources consuming multipliers with simple shift registers and is widely used in digital SOM implementations [6–8, 10].

All presented HW architectures have a two-level structure: a massively parallel distance processing elements (PEs) layer usually connected with hard links to a global circuit used for winner neuron search and weight update operations. This type of connection may be advantageous in small SOM networks but in large ones, the increasing linking complexity considerably limits their clock frequency and thus the overall performances usually expressed in MCUPS/MCPS (million of connections and updates per second respectively). Manalakos et al. proposed in [10] a parallel HW SOM systolic architecture design in which an input vector traverses all neurons in a pipelined manner, forming that way shorter links and thus a faster HW.

The lack of flexibility of hardware SOMs, which is mainly due to the point-to-point communication between neurons and especially in large SOM networks, can be overcome with the use of a Network-on-chip (NoC). NoCs are presented as an alternative to traditional shared bus allowing the connection of several PEs on a single chip [11, 12]. They enjoy an explicit parallelism, high bandwidth and a high degree of modularity, which makes them very suitable for distributed architectures such as SOM networks.

3 Proposed Architecture

3.1 Self-Organizing Map (SOM)

The architecture of a SOM can be described with a two-dimensional distribution of $L \times K$ neurons. Each neuron has a weight vector $\overrightarrow{m}$ of dimension D, which is continuously compared to the input vector $\overrightarrow{X}$:

$$\overrightarrow{X} = \{\xi_1, \xi_2, \ldots, \xi_D\} \epsilon \Re^D \tag{1}$$

Each neuron calculates the distance between its weights $\overrightarrow{m}_{l,k}$ ($0 \leq l \leq L-1, 0 \leq k \leq K-1$) and the input vector $\overrightarrow{X}$. In general, the calculated distance is the Euclidean distance:

$$\left\| \overrightarrow{X} - \overrightarrow{m}_{l,k} \right\| = \sqrt{(\xi_1 - \mu_{l,k1})^2 + (\xi_2 - \mu_{l,k2})^2 + \cdots + (\xi_D - \mu_{l,kD})^2} \tag{2}$$

Therefore, the winner neuron, which has the vector $\overrightarrow{m}_c$ closest to the input vector $\overrightarrow{X}$, is identified.

$$\mathbf{c} = \underset{l,k}{\mathbf{argmin}} \left\| \overrightarrow{X} - \overrightarrow{m}_{l,k} \right\| \tag{3}$$

During the learning phase, the winner's weights and the weights of the neurons in its vicinity are updated as described by the following equation:

$$\overrightarrow{m}_{l,k}(t+1) = \overrightarrow{m}_{l,k}(t) + \mathbf{h}_{c,l,k}(t) \left[\overrightarrow{X}(t) - \overrightarrow{m}_{l,k}(t) \right] \tag{4}$$

where $h_{c,l,k}(t)$ is the neighborhood function defined as follows:

$$\mathbf{h}_{c,l,k}(t) = \alpha(t) \times \exp\left(-\frac{\left\| \overrightarrow{r}_c - \overrightarrow{r}_{l,k} \right\|}{2\sigma^2(t)}\right) \tag{5}$$

With $\alpha(t)$ learning rate; $\sigma(t)$ Neighbourhood rate; $\overrightarrow{r}_c$ position of the winning neuron; $\overrightarrow{r}_{l,k}$ position of the neuron with index (l, k).

3.2 Network on Chip (NoC)

The structure of a 2D mesh NoC is shown in Fig. 1a. The packets are transported from a source to a target through a network of routers and interconnection channels (Link). The network is composed of processing elements (PEs) and routers. Each router is associated to a PE via a network interface whose primary function is to pack (before sending) and unpack (after receiving) data exchanged between PEs. Figure 1d illustrates the structure of packets circulating in the network using the wormhole switching technique. Each packet is composed of flits: header (opening the communication and "showing" the route to other flits), body (containing the data) and tail flit(closing the communication). The router (see Fig. 1b) is composed of a crossbar which establishes multiple links between inputs and outputs of the router according to the predefined routing algorithm and scheduling policy. The crossbar and the arbitration of packets in the router are handled with a Control

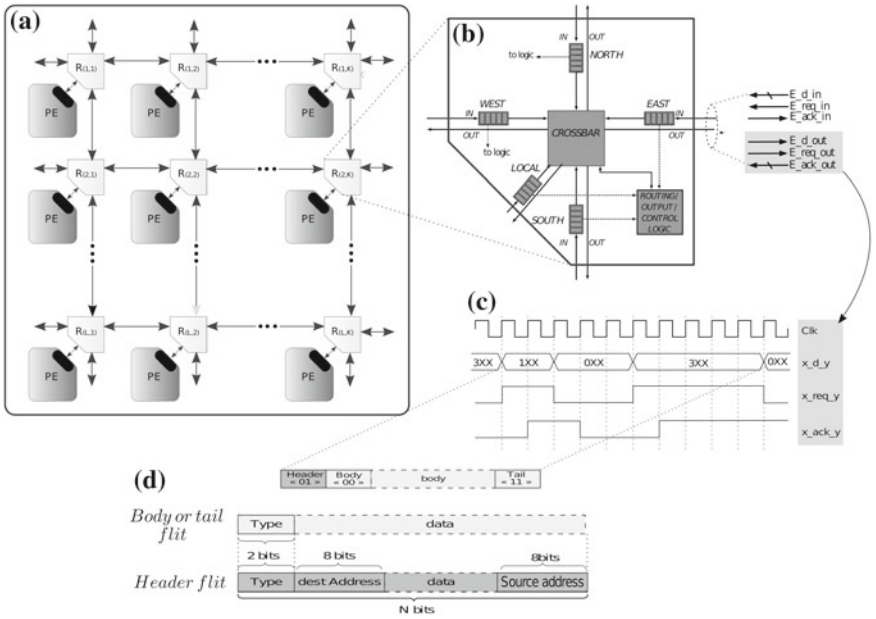

Fig. 1 **a** Structure of a 2D mesh NoC. **b** NoC router architecture. **c** Alternate bit protocol. **d** Message structure

Logic Block (CLB). Each router has input and/or output buffers, whose role is to temporarily accept flits before their transmission to either the local PE or to one of the neighboring routers. Figure 1b shows the interconnection signals of an isolated router. The connections between neighboring routers and the local PE are carried out with six signals (3 for each transmission and reception): a bidirectional data bus, a transfer request signal and an acknowledgment signal denoted with the infixes *_d_*, *_req_* and *_ack_* respectively. Message exchange between a router and its neighbors follows the alternate bit protocol illustrated in Fig. 1c. Sending a flit through a port `x_d_y` ($x \in \{W, S, E, N\}$, $y \in \{in, out\}$) is accompanied with the request signal on the same port `x_req_y`. Upon the reception of the acknowledge signal on the same port `x_ack_y`, the sending of the next flit can be proceeded. A change in a control signal (`_req_` and `_ack_` signals) is indicated by inverting its preceding value. If the request signal has the same binary level as the acknowledge signal or vice versa, the sending or receiving of a flit is successful and the router can proceed to the next one. Otherwise, it is blocked. It should be noted that the sending or receiving of a flit consumes 2 clock cycles.

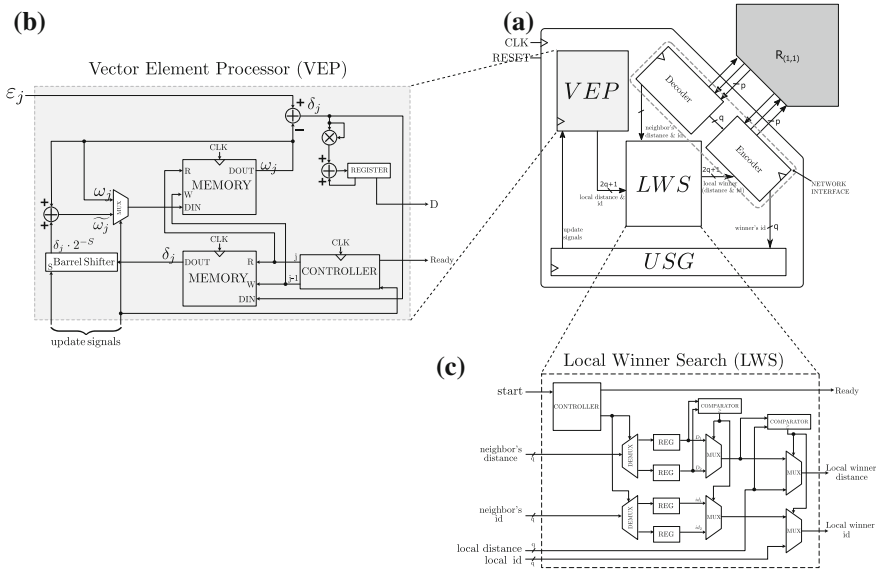

Fig. 2 **a**Architectures of the SOM-NoC's PE, **b** VEP and **c** LWS circuit

3.3 SOM-NoC

The PEs of a $L \times K$ NoC were adapted for SOM computation. The architecture of a SOM-NoC's PE is presented in Fig. 2a. It consists of 3 circuits: a Vector Element Processing (VEP) whose role is to calculate the distance and to update the weights of the corresponding neuron during the adaptation phase (see Fig. 2b); a Local Winner Search (LWS) circuit presented in Fig. 2c which carries out the comparison of the local distance and the received neighbor's distances; an Update Signal Generator (USG) is the circuit preparing update signals during the adaptation phase; and a Network Interface (NI) ensuring the sending to and receiving of data from neighboring PEs (neurons). Each PE has an identity (its address in the network) which determines the instructions its NI needs to execute during the winner search operation. The top left and the bottom right PEs have some additional functions: they initiate the winner search operation and winner id diffusion respectively.

Each PE behaves as a neuron: in the competition phase, it calculates the Euclidean distance between the input vector and its weights and send it through the NI to the nearest neighbors. The distance is propagated through the network in a systolic manner as presented in Fig. 3. Each computed distance crosses two neighboring routers before arriving at the PE's node. Upon reception of the neighboring PEs' distances, each PE compares them (with the LWS circuit) to the local one to locally determine the identity of the winner neuron. The NI is in charge to send the locally determined minimum distance and the corresponding neuron's id to the neighboring PEs. At the bottom right PE node of the network, the identity of the wining neuron is

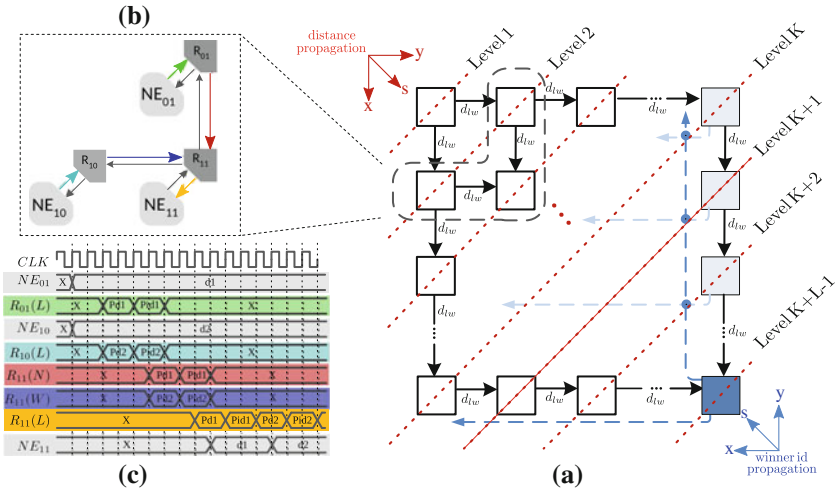

Fig. 3 Systolic architecture of SOM

known (see Fig. 3a). This identity is then broadcast to all nodes of the network while the winner neuron, as well as its neighbors, start the update of their weights. In this HW architecture, the used neighborhood function is a simple shift function carried out with a barrel shifter.

The execution time of the learning operation T_i is calculated using the competition and adaptation times T_c and T_a respectively:

$$T_i = T_c + T_a \tag{6}$$

The distance calculation is carried out simultaneously in all PEs. Then, the propagation and comparison at the PE's level are also conducted in parallel as presented in Fig. 3c. From Fig. 3, it can be seen that the time needed to send (or propagate) a calculated distance from one PE to its neighbor PE is equal to the time to cross 2 routers in the network (2 hops). Therefore, the time needed for the competition phase is given by the expression:

$$T_c = T_{cd} + \{(T_p + T_{cmp}) \times (N_{stg} - 1)\} \tag{7}$$

where T_{cd}, T_p, T_{cmp} and N_{stg} are distance calculation, propagation and comparison time and the level number respectively.

The calculation of the Euclidean distance between the input and the weight vector is done sequentially element-by-element using the VEP shown in Fig. 2a. Each elementary operation is performed in a single clock cycle. The intermediary results are stored in the accumulator, while the final results of the adaptation phase are stored in the local memory for further reuse. As it is presented in Fig. 4, each element of the input vector is processed in a single clock cycle and the final distance is ready after an additional clock cycle. The distance calculation time is given by:

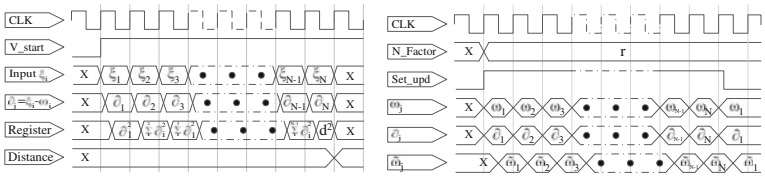

Fig. 4 Timings representing distance calculation (*left*) and update of weights (*right*)

$$T_{cd} = (N + 1) \times T_{CLK} \tag{8}$$

where N is the input vector size and T_{CLK} is the clock period.

The propagation time of a flit between two PEs in the presented NoC without collision (2 flits demanding the same output channel) is calculated by:

$$T_p = T_{pk} + \{T_r \times N_r\} + T_{dpk} \tag{9}$$

where T_{pk}, T_{dpk} are packing and unpacking time, T_r is the router latency and N_r is the number of routers to the final destination.

The adaptation phase involves two steps. Broadcasting of the identity of the winner neuron is ensured by neurons located at the end of each row of the network. The necessary broadcasting time is determined by the equation:

$$T_{bc} = (T_{pk} + T_{dpk} + 1) \times (K + L - 2) + 4T_r \tag{10}$$

Once the identity of the winner neuron is available, the USG circuit takes into account its position and generates the update signals. If a PE is concerned with the update signals, its VEP circuit starts the update phase in $T_{upd} = N \cdot T_{CLK}$ clock cycles. Figure 4 presents the timings of the update phase. The update of a weight vector element takes one clock cycle.

4 Results and Discussion

A 5 × 5 SOM-NoC using 32-element input vectors was used for performance evaluation. The parameters of the implemented architecture are presented in Table 1. The circuit was synthesized on a Xilinx Virtex-6 FPGA board. The obtained maximum working frequency is 200 MHz.

The performances of each step as well as the overall performances of the architecture are shown in Table 1.

Table 1 Parameters and performances of the proposed SOM-NoC architecture

Parameter	Value	Description
$L \times K$	5×5	SOM size
N	32	Vector size
N_{stg}	9	Level number
p	12	Data size
q	34	Flit size
R	5	Neighbour radius
T_r	3	Router latency
T_{pk}	2	Packing latency
T_{dpk}	2	Unpacking latency
Description	Step equation (clock cycles)	Time (ns)
1 Distance calculation	$T_{cd} = N + 1$	165
2 Global winner search	$T_{sw} = T_p \times (N_{stg} - 1) = 13 \times (N_{stg} - 1)$	520
3 Broadcasting of winner's id	$T_{bc} = (T_{dpk} + T_{pk} + 1) \times (L + K - 2) + 4T_r$	260
4 Update weight	$T_{upd} = N$	160
5 Recall	$T_c = T_{cd} + T_{sw}$	685
6 Adaptation	$T_a = T_{bc} + T_{upd}$	420
7 Learning	$T_i = T_c + T_a$	1105
MCPS	$\frac{L \times K \times N}{Tc} \times 10^{-6}$	1168
MCUPS	$\frac{L \times K \times N}{Ti} \times 10^{-6}$	724

The distance calculation (step 1) time depends on the input vector dimension. For a 32-element input vector, it takes 33 clock cycles, which is equivalent to 165 ns for a 200 MHz working frequency. Similarly, in the weight update phase (step 4) all PEs operate in parallel, that means the number of clock cycles needed for this phase is proportional to the input vector dimension (32). On the other hand, we note that the winner neuron search (step 2) is the most time consuming step (almost 50 % of the overall learning time). This search is done sequentially in a systolic manner as explained in Sect. 3.3 and greatly depends on the network size ($L \times K$) and thus on the number of levels of propagation which is equal to $L + K - 1$. Moreover, the winner id broadcasting phase (step 3) depends on the number of lines of the network.

The performance of the presented architecture depends on the network size and the input vector dimension. Figure 5a shows the estimated performances in terms of MCPS and MCUPS as a function of the network size for a 32-element input vector. It can be seen that the performances increase non-linearly with the network size. As the network size increases and accordingly the number of available neurons, the communication time becomes preponderant to the computation one thus limiting the

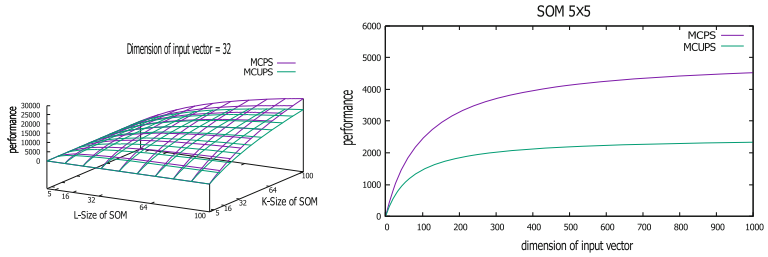

Fig. 5 **a** Performances of a SOM-NoC as a function of the network size (*left*) and **b** of a 5×5 SOM-NoC as a function of the input vector dimension (*right*)

increase in performances with regard to the number of neurons. On the other hand, for the same network size while increasing the input vector dimension, the computation becomes dominant over the communication and a rapid growth of performances as presented in Fig. 5b can be observed. Moreover, we also note that for the input vector dimension above 400 elements, the performances have a much slower rate of growth which can be explained with a very time consuming distance calculation and weight update phases which are both done sequentially on the input vector. If the calculating distance and update time exceed the time needed for the competition and broadcasting steps, there is no significant performance gain with the input vector dimension increase. Table 2 shows the comparison of the proposed HW architecture with the results reported in [7, 8, 10]. In order to make these results comparable, we presented the estimated performances (based on data from Table 1) of our architecture for a 16×16 SOM using 2048-element input vectors.

Table 2 Performance comparison

Work	Size	Input vector	Communication	Architecture	Frequency	MCUPS	Scalability
Lachmair et al. [8]	6050	194	Bus	Software core i7	NA	1628	Yes
Lachmair et al. [8]	6050	194	Bus	SIMD gNBXe processor	NA	20604	No
Hikawa and Maeda [7]	16×16	3	P to P	Parallel FPGA	33 MHz	25344	No
Manolakos and Logaras [10]	100	2048	P to P	Systolic array FPGA	148 MHz	3467	No
This work	16×16	2048	NoC	Sequential systolic FPGA	200 MHz	22555	Yes

5 Conclusion and Perspectives

We presented in this paper an FPGA implementation of a Network-on-Chip-based hardware Self-Organizing Map. Each neuron is associated to one NoC router allowing it to exchange data with other neurons of the network during both the learning and recall phases. The presented architecture is highly scalable and flexible and can be easily adapted to a large variety of applications demanding different working parameters. The implemented neurons support different input vector dimensions. The most time consuming phase is the winner search phase which is also done sequentially. The proposed architecture (SOM and NoC) is described in VHDL and its performances are evaluated for different network and input vector sizes. It has been showed that the presented architecture in its current state is most suitable for large input vector dimensions where communication time is neglected with regard to the computation one. The performance improvement of an order of magnitude in the recall phase can easily be obtained by exploring the architecture pipelining or by using faster NoC routers.

References

1. Kohonen, T.: Self-organizing map (2001)
2. Kolinummi, P., et al.: Parallel implementation of SOM on the partial tree shape neurocomputer. Neural Process. Lett. **12**(2), 171–182 (2000)
3. Talaska, T., et al.: Analog programmable distance calculation circuit for winner takes all neural network realized in the cmos technology (2015)
4. Abuelma'Ati, M., et al.: A reconfigurable gaussian/triangular basis functions computation circuit. Analog Integr. Circ. Sig. Process **47**(1), 53–64 (2006)
5. Kolasa, M., Długosz, R., Pedrycz, W., Szulc, M.: A programmable triangular neighborhood function for a kohonen self-organizing map implemented on chip. Neural Netw. **25**, 146–160 (2012)
6. Ramirez-Agundis, A., et al.: A HW design of a massive-parallel, modular NN-based VQ for RT video coding. Microprocess. Microsyst. **32**(1), 33–44 (2008)
7. Hikawa, H., Maeda, Y.: Improved learning performance of hardware self-organizing map using a novel neighborhood function (2015)
8. Lachmair, J., et al.: A reconfigurable neuroprocessor for self-organizing feature maps. Neurocomputing **112**, 189–199 (2013)
9. Aggarwal, C.C., et al.: On the surprising behavior of distance metrics in high dimensional space. Springer (2001)
10. Manolakos, I., Logaras, E.: High throughput systolic SOM IP core for FPGAs. In: IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007, vol. 2, pp. 11–61. IEEE (2007)
11. Benini, L., De Micheli, G.: Networks on chips: a new soc paradigm. Computer **35**(1), 70–78 (2002)
12. Dally, W., Towles, B.: Route packets, not wires: on-chip interconnection networks. In: Design Automation Conference, 2001. Proceedings, pp. 684–689 (2001)