electronics

MDPI

Article

Place-and-Route Analysis of FPGA Implementation of Nested Hardware Self-Organizing Map Architecture

Hiroomi Hikawa

Faculty of Engineering Science, Kansai University, Osaka 564-8680, Japan; hikawa@kansai-u.ac.jp

Abstract: Self-organizing map (SOM) is a type of artificial neural network that provides a nonlinear mapping from a given high-dimensional input space to a low-dimensional map of neurons for clustering. The clustering of high-dimensional vectors is too slow for SOM when implemented in software. In such cases, an application-specific hardware SOM accelerator is highly desirable. Field-programmable gate array (FPGA) implementation is a popular platform to implement hardware SOM. In our previous work, a nested hardware SOM architecture, which has a homogeneous modular structure to enhance expandability, was proposed. This paper investigates the impact of the nested hardware SOM on FPGA implementation tools that perform logic synthesis, place, and route (PAR). Experiments revealed that the nested architecture provided better results in resource usage and performance. FPGA resource usage of the nested architecture was 97.2% of that of the flat design on average. Importantly, the nested architecture operated at 10% higher clock frequencies compared to flat SOM designs. In addition, the pipeline computation was improved by increasing the pipeline stages so that it operates with a higher clock frequency. The operable clock frequency was 81 MHz, which was 21 MHz higher than its predecessor.

Keywords: self-organizing map; field-programmable gate array; logic synthesis; VHDL; clustering

Citation: Hikawa, H. Place-and-Route Analysis of FPGA Implementation of Nested Hardware Self-Organizing Map Architecture. *Electronics* **2023**, *12*, 4523. https://doi.org/10.3390/electronics12214523

Academic Editor: Valeri Mladenov

Received: 24 September 2023
Revised: 31 October 2023
Accepted: 1 November 2023
Published: 3 November 2023

1. Introduction

Clustering is the task of grouping sets of objects so that objects in the same group are more similar to each other than objects in other groups. Each group is referred to as a cluster, and this task is called clustering. Clustering results can be used for statistical data analysis that is used in many applications. Researchers and engineers have employed a variety of cluster models and clustering algorithms. Hierarchical clustering [1] was the earliest clustering method that is based on distance connectivity. The expectation-maximization algorithm [2] is one of the probability model-based approaches using statistical distributions, such as multivariate normal distributions. The k-means algorithm [3] is another popular alternative for clustering, in which each cluster is represented by a single mean vector. K-means clustering has been widely studied with various extensions and applied in a variety of substantive areas.

The self-organizing map (SOM) [4] is a type of unsupervised artificial neural network. The main feature of SOM is the nonlinear mapping of high-dimensional vectors onto a low-dimensional map of neurons, which has similar characteristics to the conventional clustering methods. With this feature, SOMs have been successfully used in a wide range of applications. Brito et al. [5] proposed a method for cluster visualization using the SOMs. Zhang et al. [6] presented a visual tracking of objects in video sequences. Gorzalczany et al. [7] applied SOM to complex cluster analysis problems. Campoy et al. [8] proposed a time-enhanced SOM and applied it to low-rate image compression.

The size of a SOM, i.e., the number of neurons, depends on the type of application, and certain applications require a large SOM. For some applications, SOM must handle high-dimensional vectors. For SOMs to process an image, the image is converted into vectors, but even a small image becomes a high-dimensional vector. Simulating SOMs

in software is efficient because its flexibility makes it possible to easily change the SOM configurations, such as the number of neurons and vector dimensions. However, the computational cost of the large SOM and high-dimensional vector computation are too high for software SOMs. In such cases, an application-specific hardware SOM accelerator is highly desirable, and various custom hardware SOMs have been proposed [9].

In our previous work, nested architecture for the SOM was proposed [10]. The architecture is based on a homogeneous modular structure to provide easy expandability. The nested hierarchical architecture consists of multiple modules, and each module is made up of sub-modules. All modules have a similar structure, which makes it easier for designers to design a large hardware SOM.

Field-programmable gate array (FPGA) is a suitable and commonly used platform for implementing hardware SOMs, and hardware description languages (HDLs) are widely used. It is recommended to divide the large HDL model into hierarchies so that the synthesis tools can run more efficiently, and the hierarchical design makes it easier to write smaller modules if the design is broken up into smaller modules [11]. In our previous work, performance analysis of the nested SOM architecture under various place-and-route (PAR) implementation strategies was not carried out. The PAR result can be evaluated by the worst negative slack (WNS) against the target clock constraint, which estimates the operation speed.

The previous work [10] was designed to do the clustering of high-dimensional vectors by processing the vector elements sequentially. The sequential computation requires many clocks to process the one input vector, which significantly reduces overall throughput. The throughput can be improved by taking advantage of the parallelism embedded in the SOM computation. Therefore, parallel computing architectures are suitable for implementing SOM. Parallel computing systems can be constructed using temporal parallelism, spatial parallelism, or both. The pipelining of calculations is one of the common approaches to speed up calculations. It exploits temporal parallelism, where the processing task is partitioned into multiple steps, and the task is parallelized by executing the partitioned steps by different computing elements sequentially. To cope with the problem, the previous SOM incorporated simple pipeline computing. The learning process of the SOM consists of a winner search and a weight vector update. This pipeline operation of the architecture made it possible to compute the winner search and the weight update concurrently.

As stated above, designing a nested SOM architecture must consider WNS or the critical path more carefully, which motivates the paper to analyze various WNS-oriented implementation strategies and to add a pipeline stage. The contributions of this paper are summarized as follows:

(1) Analysis of the nested architecture's impact on the PAR
Hardware SOMs with nested and flat architectures are described by VHDL, and their implementations are carried out by FPGA development tools. Then, their results are compared to reveal the efficiency of the nested architecture. In addition, the operating speeds of the two SOMs are compared through FPGA experiments to confirm the nested architecture efficiency.

(2) Enhancement of the pipeline computation
The higher the clock frequency, the higher the SOM's performance becomes. To allow operation at the higher clock frequency, pipeline stages are increased in the winner search and the weight update.

The remainder of this paper is organized as follows. Section 2 describes the SOM algorithm, and related hardware SOMs in the literature are briefly surveyed in Section 3. The details of the nested SOM architecture are discussed in Section 4. The results of the FPGA implementations and experiments are presented in Section 5, and the results are discussed in Section 6. Finally, the paper is concluded in Section 7.

2. Self-Organizing Map Algorithms

The SOM consists of neurons typically placed in a two-dimensional grid, and all neurons include weight vectors. Neuron-k includes a D-dimensional weight vector $\vec{m}_k$, which is made up of vector elements μ's.

$$\vec{m}_k = \{\mu_{k,0}, \mu_{k,1}, \cdots, \mu_{k,D-1}\} \in \Re^D \tag{1}$$

The operation of the SOM can be divided into two phases: learning and recall. In the learning phase, a map is trained with a set of training vectors. Afterward, the weight vectors of neurons are retained, and the map is used in the recall phase.

The learning phase starts with an appropriate initialization of the weight vectors. Subsequently, the input vectors, $\vec{x} \in \Re^D$, are presented to the map in multiple iterations.

$$\vec{x} = \{\xi_0, \xi_1, \cdots, \xi_{D-1}\} \in \Re^D \tag{2}$$

For each input vector, the distances to all weight vectors are calculated, and neuron-C with the shortest distance is searched. This neuron-C is called the winner neuron.

$$\begin{aligned} C &= \arg\min_k \{d_k\} \\ d_k &= ||\vec{x} - \vec{m}_k|| \end{aligned} \tag{3}$$

Euclidean distance was originally used for the vector distance d_k, but its hardware cost is too high. Thus, many hardware SOMs utilize the following Manhattan distance [9].

$$d_k = \sum_{i=0}^{D-1} |\xi_i - \mu_{k,i}| \tag{4}$$

After the winning neuron is determined, the weight vectors of neurons in the vicinity of the winner neuron-C are adjusted toward the input vector.

$$\vec{m}_k(t+1) = \vec{m}_k(t) + h(c,k,t)\{\vec{x}(t) - \vec{m}_k(t)\}. \tag{5}$$

Here, $h(c,k,t)$ is called the neighborhood function and is defined as follows:

$$\begin{aligned} h(c,k,t) &= \alpha(t) \exp\left(-\frac{d_{Ck}}{2\sigma^2(t)}\right) \\ d_{Ck} &= ||\vec{r}_C - \vec{r}_k|| \end{aligned} \tag{6}$$

where $\alpha(t)$ is the learning rate and $\sigma(t)$ defines the size of the neighborhood, both of which decrease as the learning progresses (t is the number of learning iterations). $\vec{r}_C \in \Re^2$ and $\vec{r}_k \in \Re^2$ are position vectors of the winner neuron-C and neuron-k, respectively. With this neighborhood function, vectors that are close to each other in the input space are also represented closer to each other on the map. This topology-preserving nature is a very important feature of the SOM. Like the Euclidean distance computation, the neighborhood function in Equation (6) is too complicated for hardware implementation, so many hardware SOM utilize simplified neighborhood functions such as negative powers of two or a step function [9].

3. Related Work

The researchers implemented the SOM algorithm on parallel computing hardware consisting of many processing elements (PEs). Such parallel architectures include those that consist of PEs that operate in a single-instruction-multiple-data (SIMD) manner. Porrmann et al. [12] implemented a SOM on the universal rapid prototyping system RAPTOR2000, which consists of SIMD PEs. Multiple neurons could be implemented in each PE. The RAPTOR2000 system consisted of Xilinx Virtex FPGAs. Then, Porrmann et al. [13] pro-

posed another hardware accelerator. They developed the NBX neuro processor with CMOS technology, which was made up of eight PEs and worked in SIMD mode, and the Multi-processor System for Neural Applications (MoNA) was developed by integrating the NBX neuro processors. The original SOM algorithm was modified to facilitate implementation in parallel hardware. Specifically, the Manhattan distance was used. Lachmair et al. [14] proposed gNBXe, a hardware SOM based on a modular architecture. This work was based on the principles of NBX [12]. It consisted of a global controller (GC) and the PEs that executed the neurons. The system could be easily extended by adding the gNBXe modules to the system bus. The instruction set of the PE was modified to implement Conscience SOM (CSOM).

Array processor is another parallel processing system in which the PEs are placed in an array structure. The array processor is suitable for the implementation of the SOM algorithm, too. Ramirez-Agundis et al. [15] proposed a modular, massively parallel hardware SOM. The hardware architecture was divided into a processing unit array, an address generator, and a control unit. The processing unit array was distributed into modules, with 16 units each and a maximum of 16 modules (up to 256 neurons). The proposed system was applied to a vector quantizer for real-time video coding. Tamukoh et. al. [16] proposed a winner search circuit, which was a modified version of bit-serial comparison. The proposed winner search performed a rough comparison in the early stage and a strict comparison in the later stage, which provided faster learning for massively parallel SOM architectures.

Kung [17] proposed the systolic array, which is a homogeneous network of PEs. The PEs communicate only with their neighboring PEs to receive and deliver the computing data. As its name suggests, the propagation of data through a systolic array resembles the pulse of the human circulatory system. Like other parallel computing systems, the systolic array can be used to implement the hardware SOM. Manolakos et al. [18] designed a modular SOM systolic architecture. The proposed array was made up of two types of PEs: the recall mode PE (PER) and the weights update PE (PEU). The proposed SOM was implemented on an FPGA, and a real-time vector classification problem was performed. They also developed a soft IP core in synthesizable VHDL, where the network size, vector dimension, etc. were configured by parameters. Jovanovic et al. [19] implemented a hardware SOM that used network-on-chip (NoC) communication. NoC is a network-based communication subsystem on an integrated circuit. Its key feature is that the architecture can perform different applications in a time-sharing manner by dynamically defining the use of neurons and their interconnections. The proposed hardware SOM architecture also utilized the systolic way of data exchange through the NoC, which provided high flexibility and scalability. Ben Khalifa et al. [20] proposed a modular SOM architecture called systolic-SOM (SSOM). The proposed architecture is based on the use of a generic model inspired by systolic movement and was formed by two levels of nested parallelism of neurons and connections. The systolic architecture was based on a concept in which a single data path traversed all neural PEs and was extensively pipelined.

In the hardware SOM, the winner search is the key process that governs overall SOM's computing speed because it must compare all vector distances of the neurons to find the smallest one. Researchers focused on effective methods to implement the winner search. de Abreu de Sousa et al. [21] compared three different FPGA hardware architectures—centralized, distributed, and hybrid—for executing SOM learning and recall phases. The centralized architecture used a central control unit that found the global winner and controlled all computation units. In the distributed architecture, local winners of neuron groups were identified, and the global winner was selected from them. The hybrid model combined the two architectures. Three architectures were compared in terms of chip area occupation and maximum operating frequency, and the results revealed that the centralized model outperformed the other models. All Winner-SOM (AW-SOM) proposed by Cardarilli et al. [22] did not require the identification of the winner neuron, which is a crucial operation in terms of propagation delay. Experimental results showed that if the neurons were initialized using a uniform or random distribution, the results

of AW-SOM and traditional SOM clustering were comparable in 92% of the cases. The absence of the comparator tree for the winner neuron selection considerably improved the system performance.

Various applications were applied to the hardware SOMs in the literature, such as image enlargement [16], image compression [15,20,22] and image clustering [10]. Applications other than image processing were applied to the hardware SOMs, too. A hardware SOM called SOMprocessor was proposed by de Sousa et al. [23], which used two different computational strategies to improve the data flow and flexibility to implement different network topologies. The first improvement was achieved by implementing multiplex components, which supported alternating processing of neuron sets by the arithmetic circuits. This strategy enables more flexible use of the chip so that larger networks can be processed in low-density FPGAs. The second improvement was the inclusion of a pipeline architecture for the training algorithm so that different parts of the circuit could process data at the same time. SOMprocessor was used to categorize videos of human actions for autonomous surveillance. Quadrature amplitude modulation (QAM) has been applied in many communication systems. Abreu de Sousa et al. [24] proposed an FPGA-based hardware SOM to detect 64-QAM symbols. The use of a SOM in Quadrature/In-phase pair detection enabled the QAM constellation to be adjusted continuously with no supervision. Thus, bandwidth could be saved, and training data retransmission was no longer necessary. Tanaka et al. [25] developed a single-layer deep SOM network and a fully connected neural network (FCNN). Hardware for the deep SOM network and the FCNN were designed and implemented on an FPGA. Then, an amygdala model, which is an area of the brain associated with classical fear conditioning, was implemented and applied to a robot waiter task in a restaurant.

4. Hardware SOM Architecture with Nested Structure

4.1. Nested Architecture

The nested SOM, including $M \times M$ neurons, consists of a single $M \times M$ module. The module's structure is hierarchical; thus, the $M \times M$ module is made up of four $M/2 \times M/2$ modules, and the modules in the lowest layer consist of four neurons [10]. The number of neurons can be extended by adding a new layer. The number of neurons is quadrupled because the additional layer has four sub-modules. The modules in all layers are essentially the same in their configuration; this homogeneous modular structure makes it easier for designers to write a new module using VHDL. Figure 1A shows an example of the nested SOM with 256 (16×16) neurons, which is made up of a single 16×16 module. As depicted in Figure 1B, the 16×16 module is made up of four 8×8 modules, each of which contains four 4×4 modules. Likewise, the 4×4 module consists of four 2×2 modules, and the 2×2 module contains four neurons.

The nested SOM targets high-dimensional vectors by sequentially processing vector elements. Two input signals (x_S and x_U) feed two vector elements of $\vec{x}(n-1)$ and $\vec{x}(n)$, respectively. $\xi_i(n)$ denotes the i'th vector element of n'th vector $\vec{x}(n)$. Those buses provide the SOM with pipeline computation capability, where the winner search and weight update are carried out concurrently.

Each module outputs $\vec{r}_k$ and d_k, where $\vec{r}_k$ is the position vector of the local winner neuron and d_k is the minimum vector distance of that neuron. The minimum search circuit compares the four vector distances (d_0, d_1, d_2, d_3) to determine the shortest one d_L and the corresponding winner's position vector $\vec{r}_L$. Then d_L and $\vec{r}_k$ are sent to the upper layer. Figure 2 shows the minimum search circuit, which is based on a binary search algorithm. A series of comp2 circuits selects the winner candidates in a tournament format. The centralized architecture used a central global search unit to select the global winner. This needed to be redesigned when the number of neurons increased [26]. In the nested architecture, SOM can be extended without modifying the winner search circuit because the winner search is distributed among the modules. Because the 16×16 module in Figure 1A is the top module, its local winner is the global winner. Therefore, $\vec{r}_L$ is fed back to the

sub-modules as the global winner's position vector $\vec{r}_C$. $\vec{r}_C$ is used for all neurons to compute the distance d_{Ck} to the winner neuron, which is then used to update their weight vectors. Signals A and R are control signals for the neighborhood function. The signal A determines the magnitude of the function, and R governs the radius of the neighbors. Those parameters are discussed later.

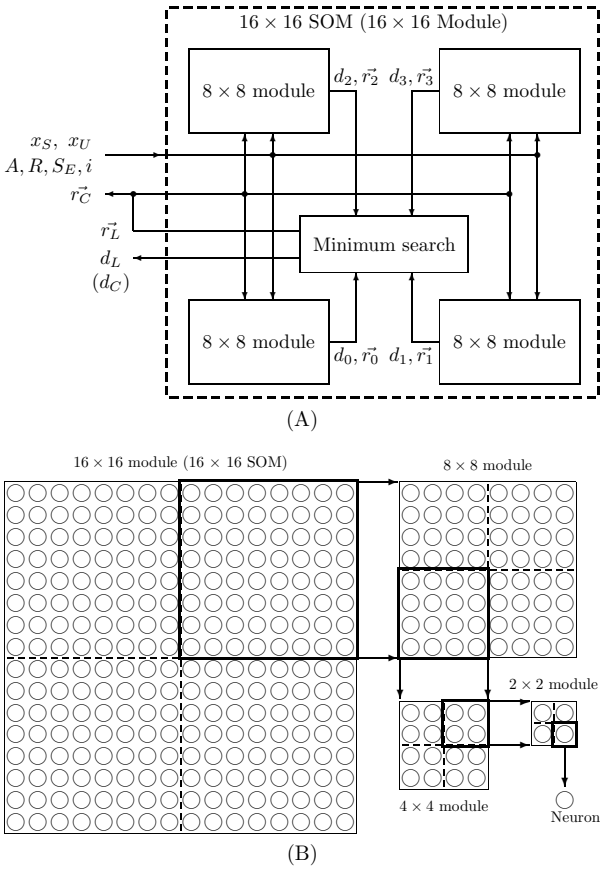

Figure 1. 16 × 16 SOM with nested architecture, (**A**) Top module, (**B**) Modules in SOM.

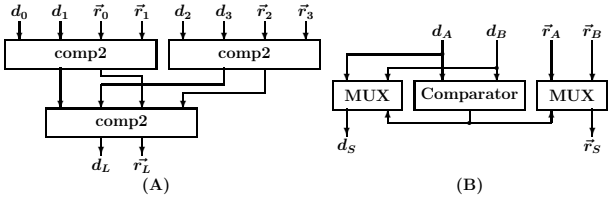

Figure 2. Minimum search circuit, (**A**) 4-input minimum circuit for SOM module, (**B**) comp2 circuit.

4.2. Neuron

As shown in Figure 3, the neuron consists of a vector distance computing unit, a weight update unit, and an update control unit. Block diagrams of these units are shown in Figures 4–6, respectively. The vector distance computing unit calculates the Manhattan distance between $\vec{x}(n)$ and $\vec{m}$. $\mu_i(n)$, i.e., the weight vector element of $\vec{m}(n)$, is given from the weight update unit through signal m_S, and signal x_S carries $\xi_i(n)$. Then, the vector distance element is computed by a subtractor and an absolute circuit (ABS), and it is accumulated in register-A. Signal S_E becomes '1' when the last vector elements are processed, and the content of register-A is transferred to register-B. Consequently, the output d_k is the Manhattan distance expressed by (4) and is fed to the minimum search unit in the module. The 2 × 2 module in the bottom layer has four neurons, and the four vector distances are fed to the minimum search circuit, which determines a local winner neuron and the shortest vector distance. As discussed before, the shortest vector distance

and local winner in the module are sent to the upper layer, and the global winner is finally determined at the top module.

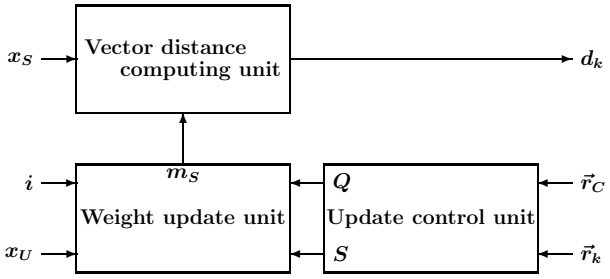

Figure 3. Neuron.

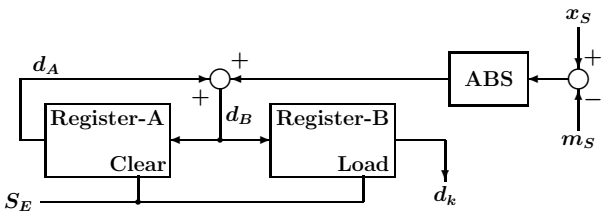

Figure 4. Vector distance computing unit.

Once the winner neuron C is determined, its position vector $\vec{r}_C = (x_C, y_C)$ is broadcast to all neurons. Then, the updated control unit of neuron k shown in Figure 5 computes position distance d_{Ck} between the winner neuron C and the neuron k.

$$d_{Ck} = ||\vec{r}_C - \vec{r}_k|| = \mid x_C - x_k \mid + \mid y_C - y_k \mid \tag{7}$$

where $\vec{r}_k = (x_k, y_k)$ is the position vector of the neuron k and d_{Ck} is the Manhattan distance between (x_k, y_k) and (x_C, y_C). Parameters Q and S, which govern the amount of the weight update, are computed from d_{Ck}.

$$Q = d_{Ck} + A \tag{8}$$

$$S = \begin{cases} 1 & \text{if } d_{Ck} \leq R \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Q and S are fed to the weight update unit shown in Figure 6. This unit consists of memory, a barrel shifter, pipeline registers, and a multiplexer (MUX). Addresses in the memory for reading and writing are specified by input ports RAdrs and WAdrs, respectively. The weight update unit updates the neuron's weight vector element. The vector element update circuit consists of a multiplexer, a barrel shifter, and memory. Weight vector elements are stored in the memory. The input vector element $\xi(n-1)$ is fed via the input signals x_U while signal m_U contains $\mu(n-1)$ from the memory. The barrel shifter shifts the difference value $\xi_i(n-1) - \mu_i(n-1)$ to the right by Q bits, thus its output value $\Delta\mu$ can be given as follows:

$$\Delta\mu = \frac{\xi(n-1) - \mu_i(n-1)}{2^Q} \tag{10}$$

The signal S from the update control unit is used as a select signal for the multiplexer. If the distance between the winner C and a neuron K is within R, then $\Delta\mu$ is added to μ in signal M_U; otherwise, no update is made. The weight vector update is represented by:

$$\mu_i(n) = \begin{cases} \mu_i(n-1) + 2^{-(d_{Ck}+A)} \cdot \{\xi(n-1) - \mu(n-1)\} \\ \qquad\qquad \text{if } d_{Ck} \leq R \\ \mu_i(n-1) \qquad\quad \text{otherwise.} \end{cases} \tag{11}$$

The signal m_T in Figure 6 feeds $\mu_i(n)$ to the memory and the pipeline register. The register output is used for the vector element distance computation in the following pipeline stage. As this equation shows, a powers-of-two-type neighborhood function is realized, in which the amount of update decreases as the distance to the winner increases as in the original neighborhood function in (6). This function can be further controlled by A and R, which change the update magnitude and the effective neighborhood size where the update is carried out. The parameters A and R correspond to $\alpha(t)$ and $\sigma(t)$ in Equation (6), respectively.

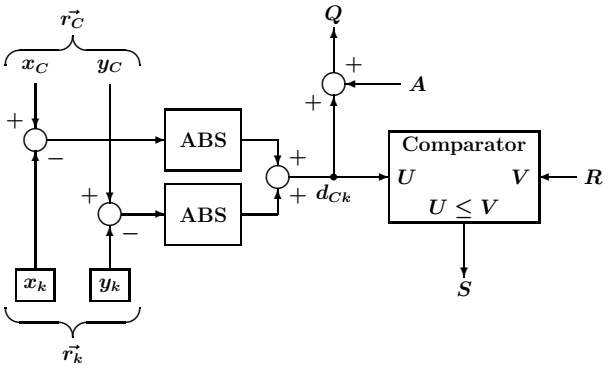

Figure 5. Update control unit.

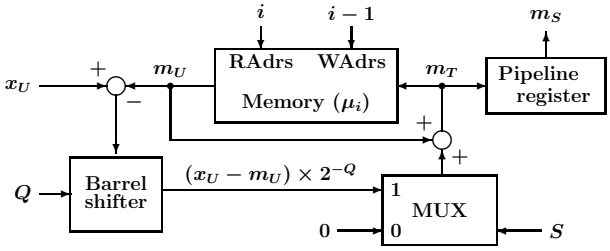

Figure 6. Weight update unit.

4.3. Pipeline Operation

The proposed architecture extends the pipeline computation proposed in [10]. Figure 7 shows the timing chart of the pipeline operation. Input vector elements of $\vec{x}(n-1)$ are given through x_U, and the weight vector elements are updated from $\vec{m}(n-1)$ to $\vec{m}(n)$ based on (5). The updated weight vector elements are immediately used to compute the vector distance to the input vector elements of $\vec{x}(n)$ given in x_S, and the winner search and weight update are carried out in the same timing.

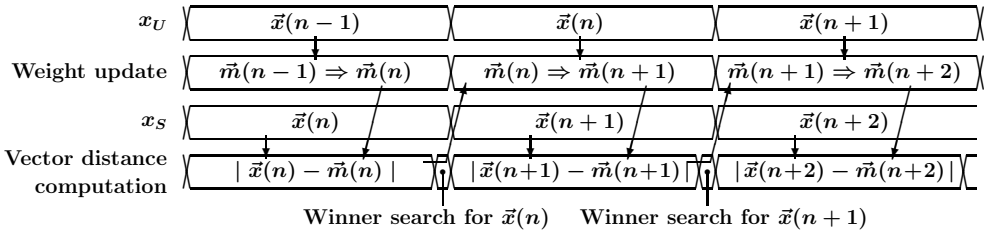

Figure 7. Pipeline processing.

A detailed timing chart is shown in Figure 8. In time slot A, weight vector element $\mu_0(n-1)$ is updated to be $\mu_0(n)$ using input vector element $\xi_0(n-1)$ in signal x_U. Here, $\mu_i(n)$ and $\xi_i(n)$ denote the i'th vector element of the n'th vectors $\vec{m}(n)$ and $\vec{x}(n)$, respectively. The updated $\mu_0(n)$ is stored in the pipeline register in the weight update unit in Figure 6 as well as the memory. The stored updated vector element $\mu_0(n)$ appears on the signal m_S in the time slot B, and is used to compute the vector element distance to $\xi_0(n)$ that is given in

the signal x_S. The distance value is added to register-A in Figure 4 and appears in signal d_A as $\hat{d}_0(n)$ at time slot C. During the same time slot, the next vector element $\mu_1(n-1)$ is updated to $\mu_1(n)$. Please note that x_U and x_S carry vector elements of the $(n-1)$'th and n'th input vectors, respectively. Meanwhile, x_U and x_S carry i'th and $(i-1)$'th vector elements, respectively. With this scheme, the vector element update and the vector element distance calculation are carried out in a two-stage pipeline. The operating clock frequency depends on the delay of the circuits for the weight update and vector element distance computation. Because the update of each vector element and the calculation of the vector element distance is performed in different time slots, the clock frequency depends only on the longer delay of the two calculation circuits. Meanwhile, in the pipeline computation proposed in [10], these two calculations were carried out in the same time slot, so the clock frequency depended on the sum of delays of the two circuits. Thus, the proposed pipeline computation is expected to allow for a higher clock frequency.

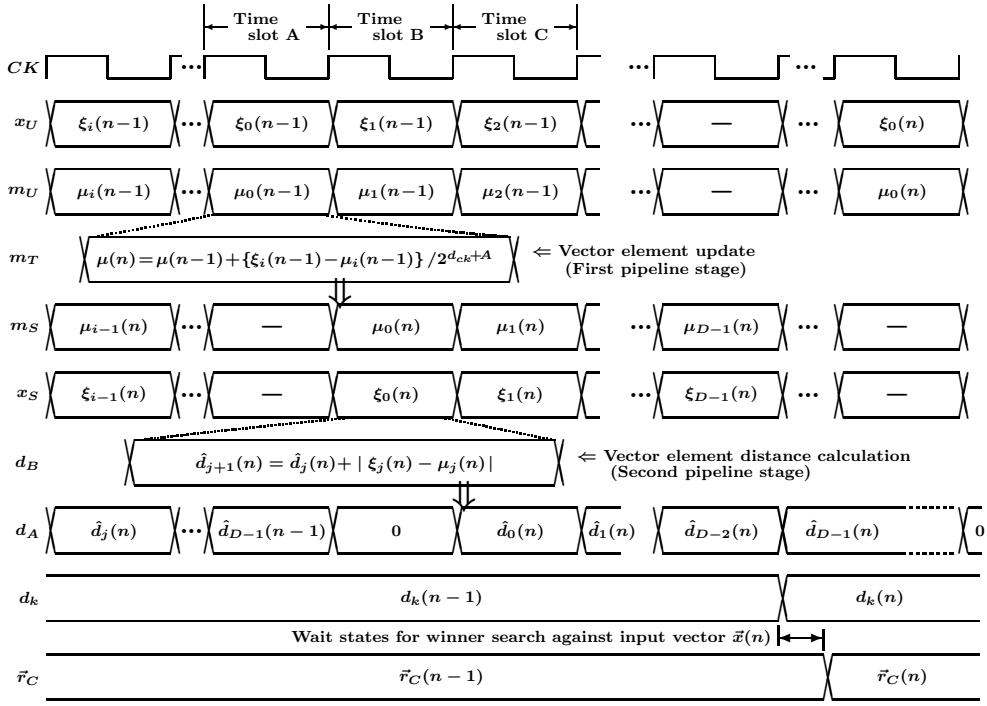

Figure 8. Detailed timing chart.

When the last vector element is processed, the vector distance d_k is determined, and the winner neuron $C(n)$ for the input vector $\vec{x}(n)$ is searched. The winner search is performed by the binary-tree search circuit distributed among the modules. Wait states are inserted for the circuit to finish the search. After the winner neuron $C(n)$ is found, its position vector $\vec{r}_C(n)$ is broadcast, and the weight vector $\vec{m}(n)$ is updated to $\vec{m}(n+1)$. In this way, the vector distance calculation and the weight vector update are carried out concurrently by the two-stage pipeline.

5. Experiments

Here, we examine the impact of the nested architecture on logic synthesis, place, and route (PAR). The non-hierarchy flat structured SOM model shown in Figure 9 was prepared for comparison. Please note that flat architectures also incorporate the same pipelined computation circuitry as the nested architecture. Both SOMs were described in VHDL, and the results of logic synthesis and PAR were compared.

The experimental configuration for testing the SOM is shown in Figure 10. The system consists of the proposed SOM, a signal generator, and dual-port memory, where N is the number of training vectors. The entire system was written using VHDL for the FPGA implementation. The signal generator generates the necessary signals for the SOM to

perform learning. The memory contains high-dimensional training vectors, and the dual-port memory is used for the pipeline computation. The precision of the vector elements in the memory was 8-bit while the weight vector elements stored in the neurons were represented by 16-bit fixed point format (8-bit integer and 8-bit fractional parts). The number of training iterations was 16 epochs, and the neighborhood function parameters A and R were implemented as follows:

$$A = 1 + E/4, \quad R = 15 - E \tag{12}$$

where E is number of epochs ($E = 0 \cdots 15$).

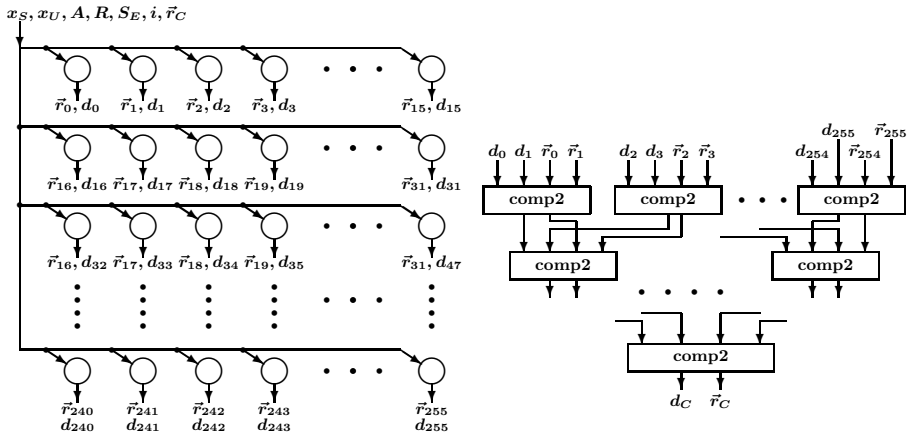

Figure 9. Flat SOM architecture.

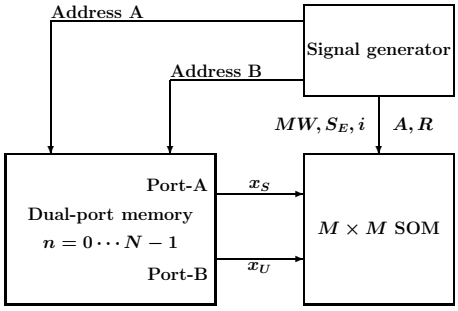

Figure 10. FPGA configuration for the learning experiment.

In the experiment, the clustering problem was applied to the SOMs. Clustering is a segmentation problem in which the objective is to gather similar data together. It can be extended to vector quantization, where a large data set can be downsized to a smaller set by substituting a prototype that represents the corresponding cluster instead of the data that belongs to the cluster. Image samples of handwritten digits taken from the Modified National Institute of Standards and Technology (MNIST) database [27] were used as training data. Each sample is a 28×28 grayscale image, making the dimension of the vector $D = 28 \times 28 = 784$. The MNIST data set consists of ten classes of handwritten digits, and 100 samples were randomly taken from each class for training. The memory shown in Figure 10 stores these training data, which is 1000 in total ($N = 1000$).

First, the nested SOM and flat SOM were described by VHDL and were processed by Quartus prime targeting Aria 10 FPGA. With clock constraints set to 20 ns (50 MHz), six implementations were performed with different strategies. The strategies in Table 1 are provided by Quartus prime. A properly selected strategy optimizes the FPGA implementation in speed, area, or power consumption. The implementation results are summarized in Table 1. The better results using the same strategy are highlighted in bold. The table shows that the nested architecture yielded faster and smaller hardware SOMs in five out of six strategies.

Table 1. Results of synthesis by Quartus Prime.

	Nested Architecture		Flat Architecture	
Implementation Strategy	**Logic Utilization (in ALMs)**	**Fmax (MHz)**	**Logic Utilization (in ALMs)**	**Fmax (MHz)**
Balanced	**40,020**	**36.21**	40,122	34.51
Performance (High effort—increases runtime)	40,358	37.82	**40,098**	**38.25**
Performance (Aggressive—increases runtime and area)	**45,045**	**37.82**	45,051	37.44
Power (High effort—increases runtime)	**40,020**	**37.47**	40,122	35.87
Power (Aggressive—increases runtime, reduces perf.)	**40,020**	**37.24**	40,122	36.27
Area (Aggressive—reduces performance)	**33,646**	**33.52**	33,831	30.46
Average	**39,851.5**	**36.68**	39,891	35.47

The same implementation experiment was carried out with AMD Vivado design tools. These tools are provided by the FPGA vendor, i.e., AMD, for users to develop their FPGA designs. Virtex-7 VC709 Evaluation Platform (xc7vx690tffg1761-2) was used for the implementation. We tested three different versions of Vivado (Vivado 2019, 2020, and 2021) for the implementations, using a 20 ns (50 MHz) clock constraint. The Vivado design tool does synthesis and implementation steps for the FPGA design. The synthesis is the process of transforming a register transfer level (RTL) design into a gate-level representation, and place-and-route the netlist onto the FPGA device resources is carried out in the implementation step. The synthesis and implementation steps include various strategies for optimizing speed, area, runtime, or power consumption. For the synthesis strategy, `Flow_PerfOptimized_high`, which optimizes the performance, was chosen. Since Vivado has many implementation strategies, 24 performance-related strategies were tested to find one that better fits SOM architecture.

The WNS was used to estimate the operation speed. Various constraints, such as clock period, are predefined to help the synthesis tools run efficiently. The tools do the PAR while trying to meet all constraints. The WNS is the difference between the target clock period that is defined in the constraint and the estimated clock period. Figure 11A shows the WNS, where labels 'a' $\sim$ 'x' on the x-axis represent the implementation strategies. Positive WNS means that the implemented design's clock period is less than the target period. Because all WNSs are negative, the closer the WNS is to zero, the higher clock frequency can be expected. This figure indicates that the speed of the circuit generated using the nested architecture was faster than that of the flat architecture for all implementation strategies. Figure 11B shows the slice utilization of the implemented designs. Slice is a basic logic element included in FPGA and constitutes a configurable logic block. Each slice contains lookup tables (LUTs), registers, a feed chain, and multiplexers. The result revealed that the nested architecture produced a smaller design.

a: Vivado Implementation Defaults,
b: Performance Explore,
c: Performance_ExplorePostRoutePhysOpt,
d: Performance_ExploreWithRemap,
e: Performance_WLBlockPlacement,
f: Performance_WLBlockPlacementFanoutOpt,
g: Performance_EarlyBlockPlacement,
h: Performance_NetDelay_high,
i: Performance_NetDelay_low,
j: Performance_Retiming,
k: Performance_ExtraTimingOpt,
l: Performance_RefinePlacement,
m: Performance_SpreadSLLs,
n: Performance_BalanceSLLs,
o: Performance_BalanceSLRs,
p: Performance_HighUtilSLRs,
q: Congestion_SpreadLogic_high,
r: Congestion_SpreadLogic_medium,
s: Congestion_SpreadLogic_low,
t: Congestion_SSI_SpreadLogic_high,
u: Congestion_SSI_SpreadLogic_low,
v: Area_Explore,
w: Area_ExploreSequential,
x: Area_ExploreWithRemap.

Next, we experimentally examined the operable highest clock frequency. Using the top five strategies that performed well on each architecture with Vivado 2021 in Figure 11, SOMs with two architectures were implemented on the FPGA, and operable clock frequencies were measured. Table 2 summarizes the operating clock frequencies and slice usages of the two SOM architectures. The larger the WNS, the higher the operating clock frequency becomes, which improves the operating speed. In terms of the circuit size, a smaller number of slices is desired. The better results using the same strategy are highlighted in bold.

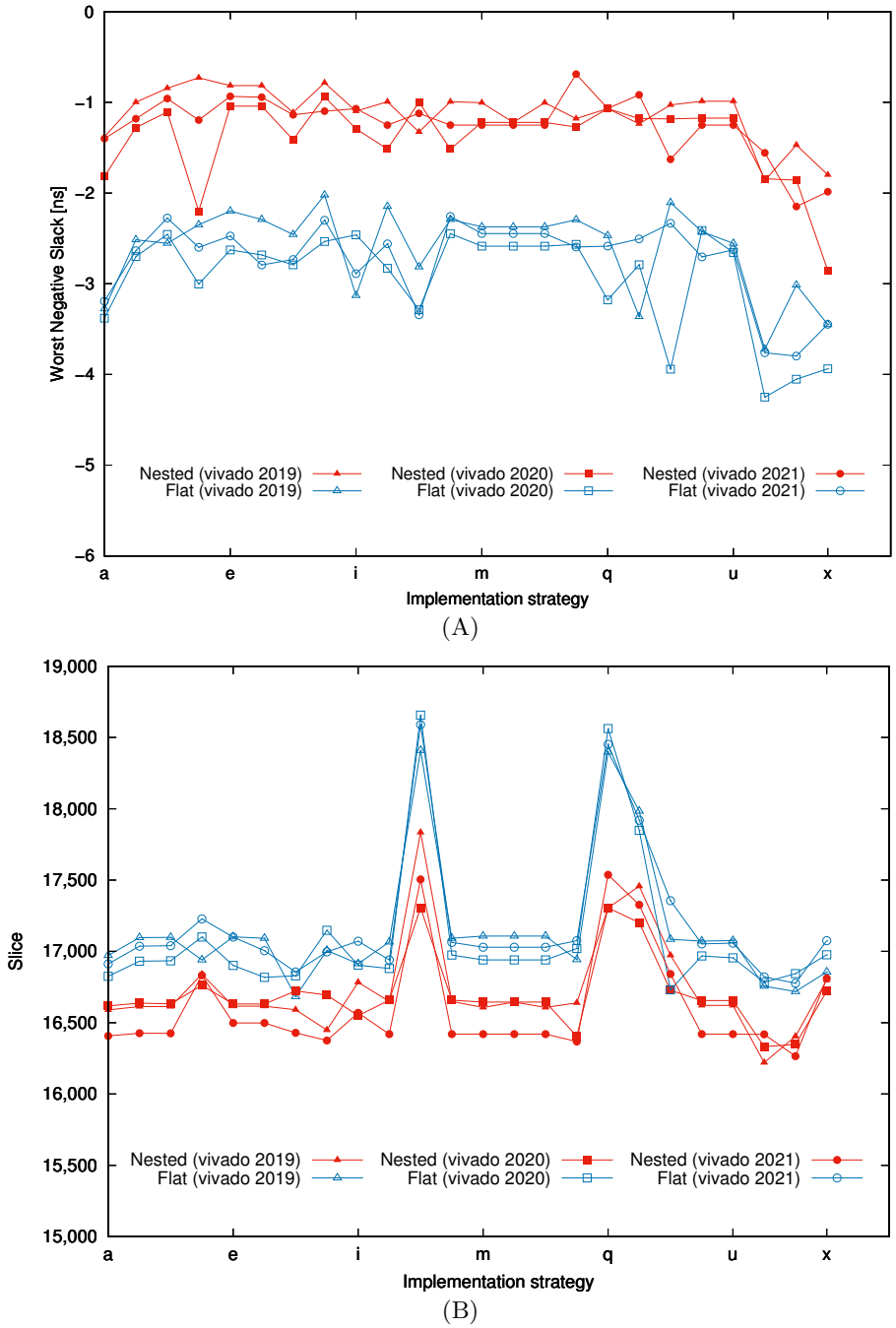

Figure 11. Logic synthesis results, (**A**) Worst negative slack (WNS), (**B**) SLICE usage.

Figure 12 shows the neuron maps after training with different clock frequencies, as well as the maps generated by the software and VHDL simulations. The neuron map is a set of visualized weight vectors that were read out from the FPGA. Until the clock frequency reached 81 MHz, the generated maps and quantization errors were the same as

those that the VHDL simulation generated, and a different map was generated at 82 MHz clock frequency. The diversity of the maps in the figure is likely due to calculation errors in the SOM circuit. Each computation circuit completes its calculation with a certain delay. Therefore, this delay must be less than the period of the clock signal. Otherwise, incorrect and incomplete calculation results will be output. For the 82 MHz clock, this timing violation appears to have occurred in the winner search circuit since the 82 MHz map is different from the 81 MHz map. Since the 83 MHz and 82 MHz maps are different, it is thought that the output results of the winner search circuit became unstable in this frequency range. Meanwhile, the vector update circuit seems to have worked correctly because the clear clustering can be observed in the map. Finally, the map collapsed at 88 MHz clock, as shown in Figure 12F because the weight update circuit apparently failed to work. The map obtained with an 82 MHz clock shows the topology-preserving nature of the SOM, and its quantization error (QE) is the same level as that of the 81 MHz map, so the SOM appears to be operating at a clock frequency above 82 MHz. Hence, we need a criterion to determine the highest operable clock frequency. Here, the operating frequency was taken as the highest frequency that produced the same map as the VHDL simulation. Therefore, 81 MHz is the highest operable clock frequency in this case.

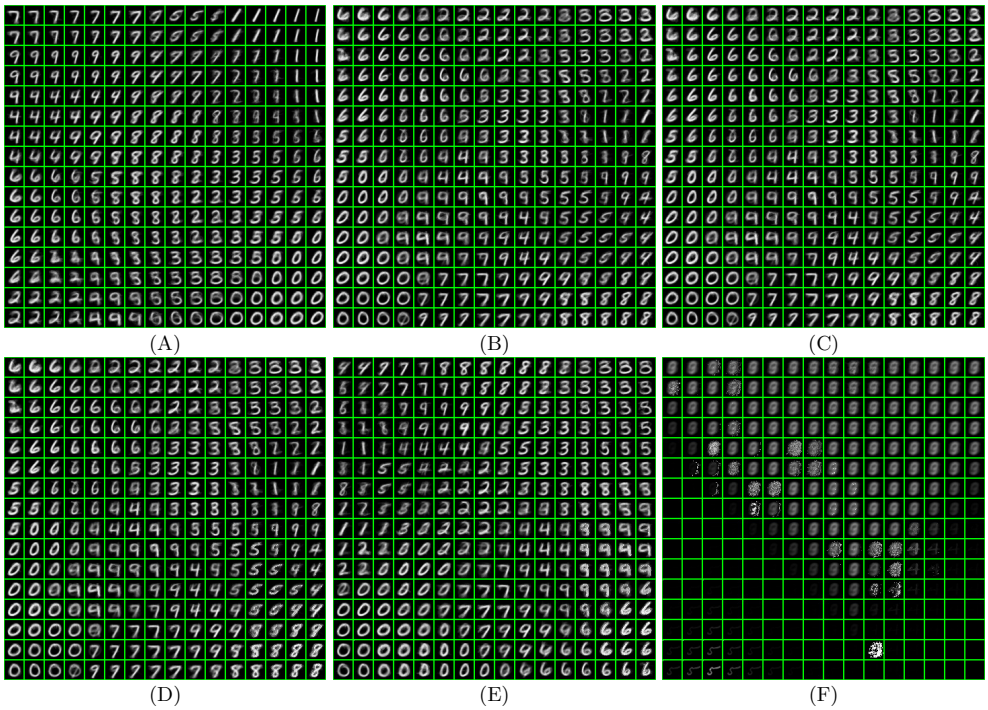

Figure 12. Neuron map after training, (**A**) Software SOM (Quantization error (QE): 1231.3), (**B**) VHDL Simulation (QE: 1683.6), (**C**) f_{CK} = 10 MHz (QE: 1683.6), (**D**) f_{CK} = 81 MHz (QE: 1522.5), (**E**) f_{CK} = 82 MHz (QE: 1507.2), (**F**) f_{CK} = 88 MHz (QE: 1811.5).

To further demonstrate the impact of the nested architecture, we implemented the nested and flat architecture SOMs on different FPGAs and compared the implementation results. The design tool processed the implementations for Virtex7, Virtex ultrascale, Kintex, Kintex ultrascale, and Zynq ultrascale FPGAs with the same constraints as the previous experiment. For synthesis and implementation strategies, `Flow_PerfOptimized_high` and `Performance explore` were used, respectively.

Table 2. Comparison between the nested SOM and flat SOM, (**A**) Top five strategies for nested architecture, (**B**) Top five strategies for flat architecture.

(**A**)						
	Proposed nested architecture			Flat architecture		
Implementation strategy	WNS (ns)	Slice	Clock (MHz)	WNS (ns)	Slice	Clock (MHz)
p:	(1) **−0.689**	**16,368**	**81**	−2.595	17,075	66
r:	(2) **−0.918**	**17,327**	**75**	−2.506	17,920	72
e:	(3) **−0.934**	**16,498**	**74**	−2.473	17,100	72
f:	(4) **−0.934**	**16,498**	**74**	−2.793	17,005	69
c:	(5) **−0.959**	**16,425**	**79**	−2.276	17,040	70
Average	**−0.887**	**16,623.2**	**76.6**	−2.529	17,228	69.8
(**B**)						
	Proposed nested architecture			Flat architecture		
Implementation strategy	WNS (ns)	Slice	Clock (MHz)	WNS (ns)	Slice	Clock (MHz)
l:	**−1.251**	**16,420**	**79**	(1) −2.258	17,063	71
c:	**−0.959**	**16,425**	**79**	(2) −2.276	17,040	70
h:	**−1.097**	**16,375**	**75**	(3) −2.300	16,997	67
s:	**−1.628**	**16,841**	**75**	(4) −2.334	17,355	67
m:	**−1.251**	**16,420**	**79**	(5) −2.447	17,029	72
Average	**−1.237**	**16,496.2**	**77.4**	−2.323	17,096.8	69.4

Table 3 summarizes the results, including WNS estimation, a lookup table (LUT), Elapsed time, and Total on-chip power estimation. For the hardware size estimation, we used the number of LUTs for the designs because some of the FPGAs are configured using a configurable logic block (CLB) instead of the Slice. Both the Slice and CLB contain the LUTs for the logic implementation. The Elapsed time is the amount of time for the tool to complete the implementation. We used a PC with an i9-12900K processor running at 3.19 GHz clock frequency. Same as the other tables, the better results are highlighted in bold. As this table shows, the nested architecture provided a smaller and faster design implementation with smaller power consumption while it required the tool to have a longer processing time.

Table 3. Nested SOM and flat SOM on various FPGAs.

Platform	**vc709**		**vcu128**		**kc705**		**kcu116**		**zcu111**	
FPGA	Virtex7 xc7vx690		Virtex ultrascale xcuv737p		Kintex7 xc7k325		Kintex ultrascale xcku5p		Zynq ultrascale xczu28dr	
Architecture	Nested	Flat	Nested	Flat	Nested	Flat	Nested	Flat	Nested	Flat
WNS (ns)	**−1.181**	−2.636	**1.905**	1.586	**−1.301**	−1.733	**4.000**	2.226	**2.036**	0.543
LUT	**56,605**	57,868	**52,364**	53,717	**55,591**	57,891	**52,472**	53,725	**53,187**	57,815
Elapsed time	**24 m 46 s**	31 m 51 s	14 m 54 s	**11 m 07 s**	37 m 06 s	**32 m 45 s**	12 m 53 s	**8 m 40 s**	13 m 36 s	**9 m 23 s**
Total on-chip power (W)	**1.611**	**1.731**	**4.052**	4.123	**1.473**	1.564	**1.378**	1.387	**2.134**	2.144

6. Discussion

As Table 2 indicates, the nested architecture resulted in a smaller and better performing SOM in all cases, even though flat architecture-friendly strategies were used. The highest

clock frequency in Table 2 is 81 MHz. The operating clock frequency of the previous research [10] was 60 MHz, but the frequency has been improved by 20 MHz. This is thought to be due to improved pipeline processing. Embedded counter counted 12,617,184 clocks during the learning. Therefore, the training time can be calculated as 0.1558 s with 81 MHz clock frequency. The total number of vector element updates during the training was 3.2113×10^9, which translates to 21.6117×10^9 vector elements were updated in one second. Since the connection update is equivalent to the vector elements update in the SOM, the 81 MHz clock frequency resulted in a learning rate of 21.61 giga connection updates per second (GCUPS).

Regarding the quantization error, the SOM implemented in the FPGA had the same quantization error at 10 MHz, but the error decreased at about 80 MHz. Since this phenomenon occurred in the higher frequency region, it is thought that small calculation errors within the circuit caused the reduction of the quantization error, as discussed before. As shown in Figure 12A shows, the quantization error of software SOM is much smaller than that of hardware SOMs implemented on the FPGA. This is because the software SOM used floating-point calculations and a traditional Gaussian neighborhood function, while all variables in the hardware SOMs are integers.

Tables 1–3, and Figure 11 clearly show that the nested architecture results in a faster and smaller design over the flat architecture. The advantage of the nested architecture is due to its structure, as both architectures incorporate the same pipeline computation. Here, from Table 2, we examine more detailed implementation results for both architectures. The average values of the characteristics of both architectures are compared.

WNS (estimated performance) $(-0.887)/(-2.323) = 0.382$

Slice (resource utilization) $166, 23.2/17, 096.8 = 0.972$

Clock freq. (physical performance) $76.6/69.4 = 1.103$

The WNS is the estimated delay difference from clock constraints (20 ns). Thus, the smaller, the better. On average, the WNS of the nested architecture is only 38.2% of the WNS of the flat design. The number of slices indicates the size of the hardware design; thus, fewer slices are desired. The nested architecture's slice usage is, on average, 97.2% of the flat design. The frequency of the clock determines the physical operating speed of the hardware SOM; thus, the clock frequency is more important than WNS. The comparison reveals that the nested architecture SOM can operate with 10% higher clock frequency compared to the flat SOM design.

Execution speeds of existing hardware SOMs in the literature are summarized in Table 4. Due to the modification to the pipeline computation method in this paper, the performance of the SOM with the MNIST training vector reached 20.6 GCUPS, which is higher than its predecessor [10] by 5 GCUPS. However, the clock frequency of the state-of-the-art SOM exceeds 100 MHz, which is much higher than this work. The proposed architecture made use of only temporal parallelism by the pipeline computing, but the SOM algorithm is made of independent tasks such as the vector distance computation and weight vector update. In addition, the vector distance computation and the vector update computation can be done with fewer clocks by processing multiple vector elements simultaneously. Therefore, the performance of the proposed architecture can be further improved by exploiting spatial parallelism, where multiple computing units are employed. For example, if the number of the vector distance computing unit and the weight update unit is four, the performance would be quadrupled. Since the cost of the spatial parallel architecture is very high, a novel implementation method would be required.

Table 4. Comparison with other hardware SOMs.

Work	SOM Size	Vector Dimension	Precision (bit)	Technology	GCUPS	Application Data Set
[13]	16 × 8	128	8	CMOS	1.32	NA
[25]	272	16 (1st) 256 (2nd)	NA	FPGA (xczu9eg)	2.80	Amygdala model
[18]	100	2048	8, 12	FPGA (xc2v6000)	3.47	NA
[12]	250 × 250	9	16	FPGA (xcv3200e)	5.70	NA
[21]	25	2	8	FPGA (xc6vlx75t)	5.85	Artificial data set
[15]	256	16	8	FPGA (xc2v6000)	6.37	Image compression
[24]	64	2	16	FPGA (u-xcku035)	8.05	QAM detection
[10]	16 × 16	784	16	FPGA (xc7vx690)	15.01	MNIST clustering
[16]	16 × 16	128	16	FPGA (xc2v6000)	17.50	Image enlargement
[14]	6050	194	16	Five FPGAs (Virtex-4FX100s)	20.60	LCVF data
[20]	16 × 16	32	8	FPGA (xc7vx485t)	24.00	Image compression
[23]	10 × 10	79	16	FPGA (xcvu440)	37.62	Video categorization
[19]	16 × 16	256	NA	FPGA (VC707)	77.39	NA
[22]	256	3	16	FPGA (xc7vx690)	**109.80**	Image compression
This work	16 × 16	784	16	FPGA (xc7vx690)	20.62	MNIST clustering

7. Conclusions

In this paper, we investigated the impact of nested hardware SOMs on FPGA implementations and improved its performance by modifying pipeline computation. The nested hardware SOM architecture has a hierarchical homogeneous modular structure that facilitates the design of larger hardware SOMs. The implementation experiments revealed that the nested architecture provided a smaller and faster design implementation with smaller power consumption while it required the tool to have a longer processing time. The FPGA resource usage of the nested architecture was 97.2% of that of the flat design, and the generated design operated at 10% higher clock frequencies compared to flat SOM designs. The original pipeline computing circuit was improved to have an additional pipeline stage, which increased its working clock frequency by 21 MHz from the previous work, and the proposed SOM reached 21.6 GCUPS.

The experimental results revealed that the quantization performance of the hardware SOM was inferior to that of the software-implemented SOM with floating-point computation, and the possibility of further parallel processing was suggested. The reduction of the quantization error is left for future research. The experimental results revealed that the architecture still needs improvement in its performance because it is inferior to other hardware SOMs in its operating speed. The parallel computation in this work only exploited temporal parallelism, omitting spatial parallelism. Spatial parallel computing is expected to improve the performance of the hardware SOM, but the required hardware cost is much higher. As another future work, we plan to improve performance by introducing spatial parallel processing, the hardware cost of which is lowered.

Funding: This research was funded by JSPS KAKENHI Grant Number JP20K11999.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Jain, A.K.; Dubes, R.C. *Algorithms for Clustering Data*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1988.
2. Dempster, A.P.; Laird, N.M.; Rubin, D.B. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J. R. Stat. Soc. Ser. B Methodol.* **1977**, *39*, 1–38.
3. Alhawarat, M.; Hegazi, M. Revisiting K-Means and Topic Modeling, a Comparison Study to Cluster Arabic Documents. *IEEE Access* **2018**, *6*, 42740–42749. [CrossRef]
4. Kohonen, T. *Self-Organizing Maps*; Springer: New York, NY, USA, 2001.
5. Brito da Silva, L.E.; Wunsch, D.C. An Information-Theoretic-Cluster Visualization for Self-Organizing Maps. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 2595–2613. [CrossRef] [PubMed]
6. Zhang, Y.; Tang, Y.; Fang, B.; Shang, Z. Real-Time Object Tracking in Video Pictures Based on Self-Organizing Map and Image Segmentation. In Proceedings of the 2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference, Chongqing, China, 20–21 December 2014; pp. 559–563. [CrossRef]
7. Gorzalczany, M.B.; Rudzinski, F. Generalized Self-Organizing Maps for Automatic Determination of the Number of Clusters and Their Multiprototypes in Cluster Analysis. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *29*, 2833–2845. [CrossRef] [PubMed]
8. Campoy, P.; Gutiérrez, P. Image Compression by a Time Enhanced Self Organizing Map. In *Progress in Pattern Recognition, Image Analysis and Applications*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4225, pp. 985–992. [CrossRef]
9. Jovanovic, S.; Hikawa, H. A Survey of Hardware Self-Organizing Maps. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 8154–8173. [CrossRef] [PubMed]
10. Hikawa, H. Nested Pipeline Hardware Self-Organizing Map for High Dimensional Vectors. In Proceedings of the 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, UK, 23–25 November 2020; pp. 1–4. [CrossRef]
11. Hamblen, J.O.; Furaman, M.D. *Rapid Prototyping of Digital Systems: A Tutorial Approach*; Kulwer Academic Publishers: Norwell, MA, USA, 2001.
12. Porrmann, M.; Franzmeier, M.; Kalte, H.; Witkowski, U.; Rückert, U. A Reconfigurable SOM Hardware Accelerator. In Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2002), Bruges, Belgium, 24–26 April 2002; pp. 337–342.
13. Porrmann, M.P.; Witkowski, U.; Rückert, U. A Massively Parallel Architecture for Self-Organizing Feature Maps. *IEEE Trans. Neural Netw.* **2003**, *14*, 1110–1121. [CrossRef] [PubMed]
14. Lachmair, J.; Merenyi, E.; Porrmann, M.; Ruckert, U. A Reconfigurable Neuroprocessor for Self-Organizing Feature Maps. *Neurocomputing* **2013**, *112*, 189–199. [CrossRef]
15. Ramirez-Agundis, A.; Gadea-Girones, R.; Colom-Palero, R. A hardware design of a massive-parallel, modular NN-based vector quantizer for real-time video coding. *Microprocess. Microsyst.* **2008**, *32*, 33–44. [CrossRef]
16. Tamukoh, H.; Aso, T.; Horio, K.; Yamakawa, T. Self-Organizing Map Hardware Accelerator System and its Application to Realtime Image Enlargement. In Proceedings of the 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541), Budapest, Hungary, 25–29 July 2004; Volume 4, pp. 2683–2687. [CrossRef]
17. Kung, S.Y. *VLSI Array Processors*; Springer: New York, NY, USA, 1998.
18. Manolakos, I.; Logaras, E. High Throughput Systolic SOM IP Core for FPGAs. In Proceedings of the 2007 IEEE International Conference on Acoustics, Speech and Signal Processing—ICASSP '07, Honolulu, HI, USA, 15–20 April 2007; Volume 2, pp. II-61–II-64. [CrossRef]
19. Jovanovic, S.; Rabah, H.; Weber, S.; Lamour, I.J.; de Lorraine, U. High Performance Scalable Hardware SOM Architecture for Real-Time Vector Quantization. In Proceedings of the 2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS), Sophia Antipolis, France, 12–14 December 2018; pp. 256–261. [CrossRef]
20. Ben Khalifa, K.; Blaiech, A.G.; Bedoui, M.H. A Novel Hardware Systolic Architecture of a Self-Organizing Map Neural Network. *Comput. Intell. Neurosci.* **2019**, *2019*, 8212867. [CrossRef] [PubMed]
21. de Abreu de Sousa, M.A.; Del-Moral-Hernandez, E. Comparison of Three FPGA Architectures for Embedded Multidimensional Categorization through Kohonen's Self-Organizing Maps. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4. [CrossRef]
22. Cardarilli, G.C.; Di Nunzio, L.; Fazzolari, R.; Re, M.; Spano, S. AW-SOM, an Algorithm for High-Speed Learning in Hardware Self-Organizing Maps. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 380–384. [CrossRef]
23. de Sousa, M.A.d.A.; Pires, R.; Del-Moral-Hernandez, E. SOMprocessor: A High Throughput FPGA-Based Architecture for Implementing Self-Organizing Maps and Its Application to Video Processing. *Neural Netw.* **2020**, *125*, 349–362. [CrossRef] [PubMed]
24. de Abreu de Sousa, M.A.; Pires, R.; Perseghini, S.D.S.; Del-Moral-Hernandez, E. An FPGA-based SOM Circuit Architecture for Online Learning of 64-QAM Data Streams. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8. [CrossRef]
25. Tanaka, Y.; Tamukoh, H. Hardware Implementation of Brain-Inspired Amygdala Model. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–5. [CrossRef]

26. Hikawa, H.; Maeda, Y. Improved Learning Performance of Hardware Self-Organizing Map Using a Novel Neighborhood Function. *IEEE Trans. Neural Netw. Learn. Syst.* **2015**, *26*, 2861–2873. [CrossRef] [PubMed]
27. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]