Reconfigurable Self-Organizing Neural Network Design and it's FPGA Implementation

Basma M. K. Younis
Technical Engineering College
In Mosul

Basil Sh. Mahmood
College of Electronics
University of Mosul

Fakhraldeen H. Ali
Computer Engineering Department
University of Mosul

Abstract

The use of Kohonen self-organizing feature maps in real time applications requires high computational performance, especially for embedded systems and hence neural network chips are essential. A digital architecture of Kohonen neural network with learning capability and on-chip adaptation and storage is proposed with the implementation of Kohonen Self-Organizing Map (SOM) neural networks on the low-cost Spartan-3 FPGAs. The architecture of this digital chip based on the idea that some assumptions for the restrictions of the algorithm can simplify the implementation. Using the Manhattan distance, a special treatment of the adaptation factor, and neighborhood functions will decrease the necessary chip area so that a high number of processing elements can be integrated on one chip.

Keywords: FPGA, Weight Vectors, Manhattan distance, Learning

التصميم المادي على رقاقة FPGA للشبكة العصبية المنظمة للخواص ذاتيا

بسمة محمد كمال
الكلية التقنية في الموصل
مؤسسة المعاهد الفنية

باسل شكر محمود
كلية هندسة الالكترونيات
جامعة الموصل

فخر الدين حامد علي
قسم هندسة الحاسبات/كلية الهندسة
جامعة الموصل

الخلاصة

إن استخدام شبكة كوهون العصبية المنظمة للخواص ذاتيا يتطلب قيامها بأداء حسابات كثيرة الأمر الذي يستدعي تنفيذ هذه الحسابات تنفيذا ماديا لا برمجيا و خاصة عندما تكون هذه الشبكة جزءا في المنظومات المطمورة . إن التصميم الرقمي المقترح لهذه الشبكة تم إسقاطه على رقاقة FPGA ثم تم فحص أداء الرقاقة الناتجة . يستطيع الكيان المادي المنفذ لهذه الشبكة تنفيذ طوري الشبكة , طور التعلم و طور الفحص , وبذلك لا نحتاج احتساب أوزان الشبكة لحل مسألة ما بوساطة حاسبة خارجية ثم خزن هذه الأوزان في ذاكرة الشبكة , و إنما تقوم الشبكة هي بحساب الأوزان في طور التعلم و بذلك نستطيع استخدام نفس الرقاقة لحل أية مسألة. ولتقليل حجم الرقاقة المطلوبة للتصميم و زيادة سرعة التنفيذ قمنا بتحوير و تقريب بعض العلاقات الرياضية كاستخدام مسافة منهاتن بدلا من المسافة الاقليدية أو استخدام الجداول الحسابية بدلا من حساب العلاقات الرياضية.

Received 27 May 2008 **Accepted 13 Aug. 2008**

1. Introduction

Artificial neural networks (ANNs) are systems based on mathematical algorithms, which are derived from the field of neuroscience and are characterized by intensive arithmetic operations [1]. These networks display interesting features such as parallelism, classification, optimization, adaptation, generalization and associative memories [2]. As artificial neural networks (ANNs) has gained popularity in a variety of application domains, it is critical that these models run fast and generate results in real time. Although a number of implementation of neural networks are available on sequential machines, most of these implementations require inordinate amount of time to train or run ANNs, especially when the ANN models are large. One approach for speeding up the implementation of ANNs is to implement them on parallel processors. Another approach is to design special hardware. So development of digital neurohardware is driven by the desire to speed-up the simulation of ANN and/or to achieve a better performance-to-cost ratio than general-purpose systems [3].

With the arrival of Field Programmable Gate Arrays (FPGAs), it has been possible to build an entire ANNs using only FPGA and memory. Even if today's FPGAs are of considerable size, each processor element (neuron) must be relatively simple if a highly parallel computer is to be constructed from them. Also, it is possible to build very powerful and efficient computers using bit-serial processing elements with SIMD (Single Instruction stream, Multiple Data streams) control. A major benefit of using FPGAs is the fact that different architectural variations can easily be tested and evaluated on real applications [4].

Among the architecture and algorithms suggested for ANNs, the Self-Organizing Map ,proposed by Kohonen, has the special property of effectively creating spatially organized "internal representation" of various features of input signals and their abstraction [5]. In this work, the processing elements of Kohonen neural network are designed and operational testes are verified through image compression application using vector quantization algorithms for reducing the transmission bit rate or the storage. The hardware organization using FPGA has been modeled and simulated in VHDL. The Xilinx family (XC3S200FT256 Spartan-3 FPGAs) is chosen as target technology [6],[7].

In section 2, the self-organization algorithm is introduced. Section 3 deals with two different designed architectures for the algorithm. Section 4 talks about the synthesis and implementation results. Section 5 gives the test results and finally, conclusions are given in Section 6.

2. Kohonen Network Self Organizing Algorithm

Self-Organizing Maps (SOMs) as proposed by Kohonen [5] use an unsupervised learning algorithm to form a nonlinear mapping of high-dimensional input space onto a low-dimensional (in most cases two-dimensional) map of neurons. SOMs have been successfully applied to a wide range of technical applications [5]. To comply with the need of high performance and low power consumption, as forced by many applications, the use of special purpose hardware is often inevitable. Especially in embedded applications a small physical size of the implementation is essential. The main steps of the algorithm are :

A. Selecting of the Best Matching Cell

Consider the two-dimensional network of cells. their arrangement can be hexagonal, rectangular, etc. Let (in matrix notation) $x = [x_1, x_2, ..., x_n]^T \in R^n$ be the input vector that, for simplicity and computation efficiency, is assumed to be connected in parallel to all the neurons *i* in this network. (Subsets of the same input signals can be connected randomly to the

cells [5]). The weight vector of cell i shall henceforth be denoted by $w_i = [w_{i1}, w_{i2}, ..., w_{in}]^T \in R^n$. The better measure for the match of x with w_i is based on the Euclidean distances between x and w_i .

$$E_i = \sqrt{\sum_{j=1}^{k} (w_i(j) - x(j))^2} \quad \text{... (1)}$$

where k is the number of elements in the input and weight vectors. The minimum distance defines the "winner" w_c.

B. Updating the Weight Vectors

It is crucial to the formation of ordered maps that the cells doing the learning are not affected independently of each other, but as topologically related subsets, on each of which a similar kind of correction is imposed. During the process, such selected subsets will then encompass different cells. The net corrections at each will thus tend to be smoothed out in the long run. An even more intriguing result from this sort of spatially correlated learning is that the weight vectors tend to attain values that are ordered along the axes of the network.

In biophysically inspired neural network models, correlated learning by spatially neighboring cells can be implemented using various kinds of lateral feedback connection and other lateral interconnections. In the present process, we want to enforce lateral interconnection directly in a general form, for arbitrary underlying network structure, by defining a neighborhood set *Nc* around a cell c. At each learning step, all the cells within *Nc* are updated, where cells outside *Nc* are left intact. This neighborhood is centered around that cell for which the best match with input x is found:

$$\|x - w_c\| = min\{ \|x - w_i\| \} \quad \text{... (2)}$$

The width or radius of *Nc* can be time-variable. In fact, for good global ordering, it is experimentally turned out to be advantageous to let *Nc* be very wide in the beginning and shrink monotonically with time (Figure 1).

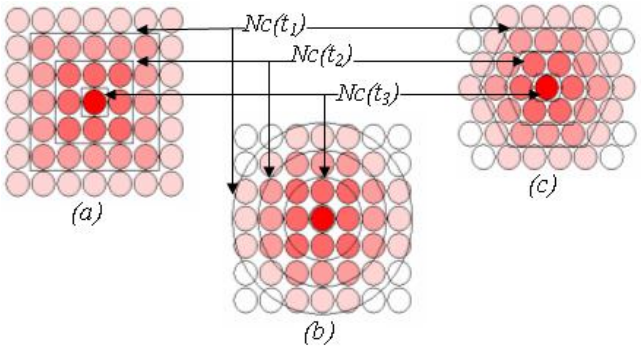


Figure 1:*Examples of Topological Neighborhood Nc(t), where $t_1<t_2<t_3$ for Different Arrangements (a)Rectangular (b)Circular and(c)Hexagonal*

The explanation for this may be that a wide initial *Nc*, corresponding to a coarse spatial resolution in the learning process, first induced a rough order in the w_i values, after which narrowing the *Nc* improves the spatial resolution of the map; the acquired global order, however, is not destroyed later on. It is even possible to end the process with *Nc*={c}. That is, finally updating the best-matching unit "winner" only, in which case the process reduces to

simple competitive learning. Before this, however, the "topological order" of the map would have to be formed. The update process in discrete time notation is:

$$w_i(t+1) = \begin{cases} w_i(t) + \alpha(t)[x(t) - w_i(t)] & if \ \ i \in Nc(t) \\ w_i(t) & if \ \ i \notin Nc(t) \end{cases} \quad ...(3)$$

where $\alpha(t)$ is a scalar value "adaptation gain" $0<\alpha(t)<1$ which is related to a similar gain used in the stochastic approximation process [5], $\alpha(t)$ should decrease with time.

An alternative notation is to introduce a scalar "kernel" function $h_{ci}=h_{ci}(t)$

$$w_i(t+1) = w_i(t) + h_{ci}(t)[x(t) - w_i(t)] \quad ...(4)$$

whereby, above, $h_{ci}(t)= \alpha(t)$ within Nc and $h_{ci}(t)=0$ outside Nc. On the other hand, the definition of h_{ci} can also be more general; a biological interaction often has the shape of "bell curve" . Denoting the coordinate of cell c and i by the vector r_c and r_i respectively, a proper form for h_{ci} might be:

$$h_{ci} = h_0 \ exp\left(\frac{-\|r_i - r_c\|^2}{\sigma^2} \right) \quad ... (5)$$

with $h_0=h_0(t)$ and $\sigma = \sigma(t)$ as suitable decreasing functions of time [5].

3. Hardware Architectures

The chip should contain a high number of elements so that large maps are possible without making the system too complex. To avoid the bottleneck between the processor element (PE) and the memory, the chip should contain on-chip-memory (local RAM) for each element. And the system should be simple enough to be extended by adding new chips (expandable).

In order to find a solution that fits all these requirements, a compromise between the functions of each element and the number of elements per chip has to be found. If the PE would have the capability of calculating the exact equations of the SOM learning rule and the distance, the necessary chip area is too large for a high number of elements per chip. On the other hand, restrictions that are made for the calculations should not change the function of self organizing feature maps.

Designs presented in this work are such a compromise, these designs are based on the idea to simplify the equations due to hardware aspects so that an efficient implementation is possible.

Simulation of self organizing feature maps can be divided into two phases. During *learning phase*, the input vector is sent to all processor elements and each PE will calculate the distance of this input vector to its

local weight vector. In most cases, the Euclidean Distance is used. After finding the PE with the minimum distance (Best Match), the adaptation factor α is calculated for all elements and the weights are updated. During *recall phase*, the new input vector is sent to all elements and the results of the calculations are the coordinates of the Best Match.

In a digital hardware implementation, a large chip area is needed to realize the square root and the multiplier functions. In order to get a smaller and faster chip design, the following restrictions have been made:

First: the Manhattan distance is used instead of the Euclidean distance. Therefore no multipliers are required to calculate the distance.

$$Manhattan\ distance_i = \sum_{j=1}^{k} |x(j) - w_i(j)| \quad \ldots(6)$$

The calculation of this distance can be realized in hardware with adders and simple gates.

Second: To calculate the weight updates, it is necessary to multiply the adaptation factor α with the difference between x and w (cf. updating the weight vectors discussed in [9]). Due to hardware aspects, it is very easy to multiply two numbers if one of these numbers is an element of the following set

$$\left\{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots, \frac{1}{2^n}\right\} \quad \ldots \quad (7)$$

This makes it possible to use shifters instead of multipliers for the adaptation.

Third: To calculate the Neighborhood functions *NC* for each iteration, a lookup table is used. This can be realized in hardware with embedded memory blocked in the chip (Block RAMs).

3.1. Full Parallel 4×4 Network

In this section, a digital hardware implementation of self organizing 4x4 map is presented. The chip has 26 pins as described in Figure 2, Eight different instructions can be executed as shown in table 1.

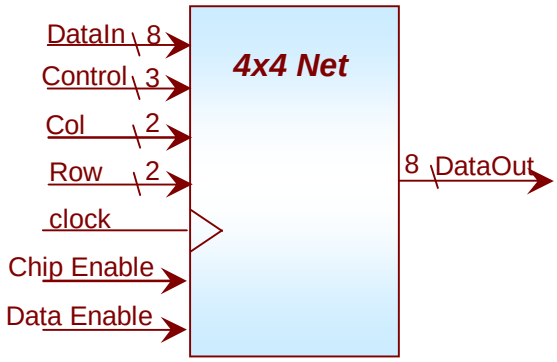


Figure 2: *4x4 Network Block Diagram*

Table 1 :*The Instruction Set of 4x4 Network*

Instruction	Description	Control Pins Value
Reset	Reset all Calculation Units	000
WriteWeight	Write Weight from DataIn to Internal RAM	001
CalDist	Calculate the distance between the input vector and the weight vectors	010
CalMinDist	Find Location of the Winner [Minimum]	011
LoadAlpha	Load the adaptation factor	100
UpdateWeight	Update Weights [the winner and surrounding]	101
ReadWeight	Read the output value	110
NOP	No operation	111

Description of the Design

The block diagram shown in figure 3 presents the internal structure of the designed network. The processing elements are working in SIMD (single instruction stream, multiple data stream) manner, controlled by an external unit. Every processing element performs the calculation required for both learning and recall. All processing elements are synchronously clocked with a single clock signal. An eight bit bus is used for data transfers to the processing elements which are addressed by the row and column lines.

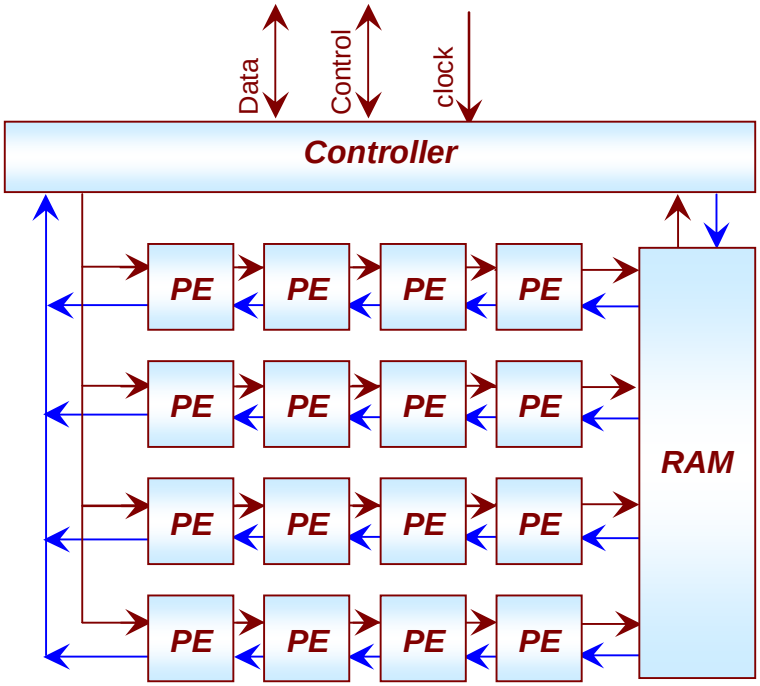


Figure 3: *4×4 Network Design Structure*

I- Processor Element Hardware Architecture

Each neuron (PE) consists of two blocks as shown in Figure 4. The weight memory block contains 16-byte (vector length) of SRAM used to store the synapse weight for that neuron while all necessary operations for the data processing are implemented in the calculation unit block which produces the adaptive weight vector elements via 8-bit bus.

A calculation will be inactive until it gets a *Reset* on the instructions data line. The components of the input vector are transmitted over the input data line and the distance to the weight vector component on the weight data line is calculated when the instruction on the instructions data line is *CalDist* see table 2.

After finding the minimum, which is described further on, adaptation is started. In order to realize the neighborhood function, a value from a lookup table (neighbor) is used to be added to the value of input (alpha) when the instruction on the instructions data line is *LoadAlpha*. Then, the new weight vector components are calculated (one component per clock cycle). This is done when the instruction on the instructions data line is *UpdateWeight*.

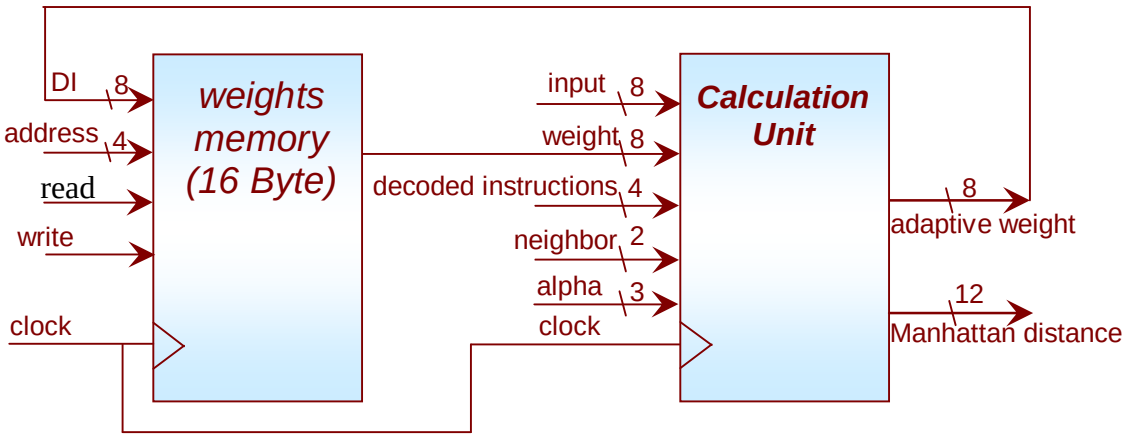


Figure 4: *One Neuron (PE) Block Diagram*

Table 2 :*The Instruction Set of the Calculation Unit*

Instruction	Code	Description	Mathematical Function performed
Reset	0001	Reset the calculation unit	All variables values equal zero
CalDist	0010	Calculate the distance between the input vector and the weight vectors	$Manhattan_i = \sum_{j=1}^{k} \lvert x(j) - w_i(j) \rvert$
LoadAlpha	0100	Load the adaptation factor	α_{reg} = neighbor + alpha
UpdateWeight	1000	Adapt the weight	adaptive weight = weight+(Input-weight)>> α

Calculation Unit Hardware Architecture

In order to perform the functions mentioned above, the calculation unit could be partitioned into the following elements:

1. The logic circuit shown in Figure 5 is used to calculate the Manhattan distance. The operation for this circuit is controlled by the *CalDist* instruction.

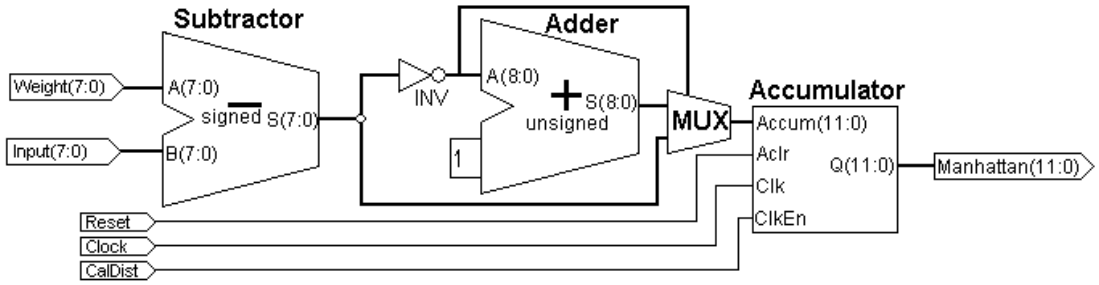


Figure 5: *Sum of Absolute Difference Circuit*

2. The logic circuit shown in Figure 6 is used to calculate the number of shifting bit positions used later in adaptation process.The operation of this circuit is controlled by the *LoadAlpha* instruction.

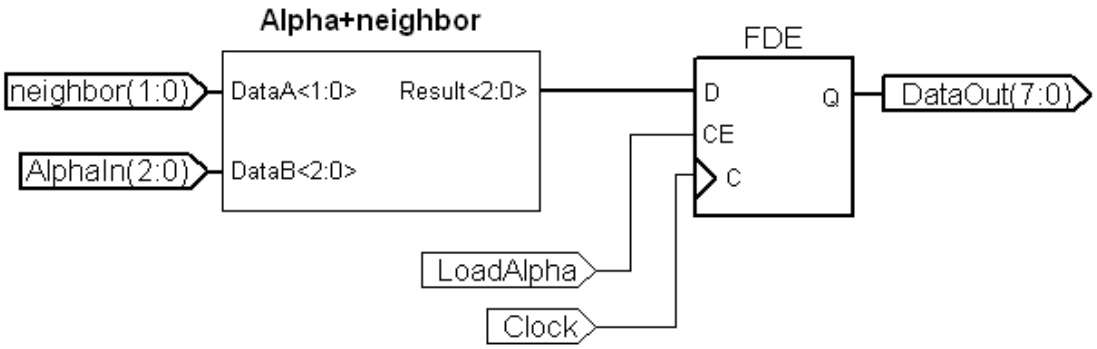


Figure 6: *Load Alpha Instruction Circuit*

3. The logic circuit shown in Figure 7 is used to calculate the updated weight values during adaptation processing. The operation for this circuit is controlled by the *UpdateWeight* instruction.

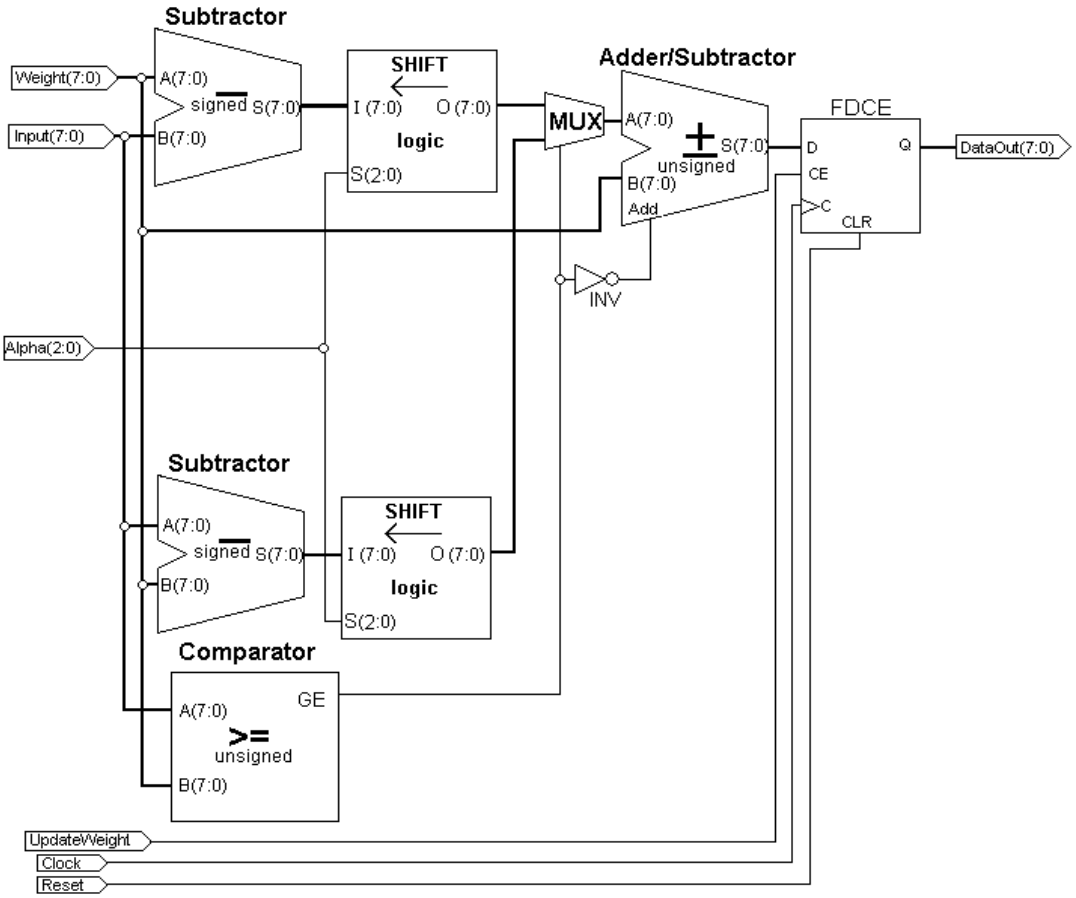
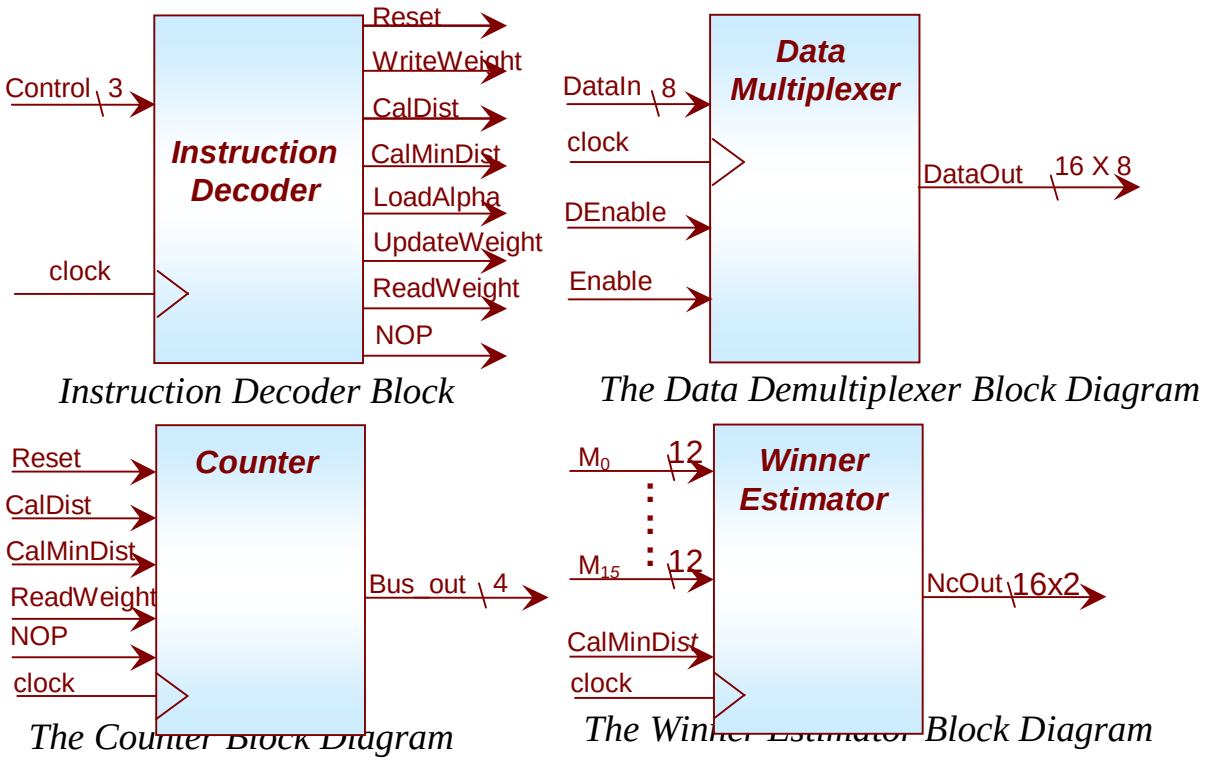


Figure 7: *Update Weight Instruction Circuit*

II- Control Unit Architecture

The control unit performs many common functions like *Instruction Decoding*, *Data Multiplexing*, Address Generating using *Counter and Winner Estimator* for finding the best matching cell during learning. Figure 8 illustrates these elements.



Instruction Decoder Block *The Data Demultiplexer Block Diagram*

The Counter Block Diagram *The Winner Estimator Block Diagram*

Figure 8: *Control Unit Elements*

3.2 Semi-Parallel 8×8 Network

In this section, a digital hardware implementation of self organizing 8×8 map is presented. The chip has 28 pins as described in Figure 9.

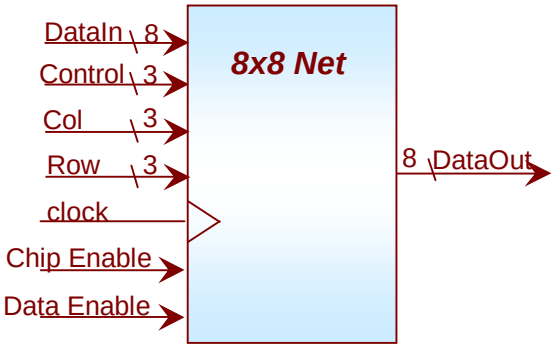


Figure 9: *8×8 Network Block Diagram*

The block diagram shown in figure 10 presents the internal structure of the designed network. Every processing element does the calculation required for learning and recall. All processing elements are synchronously clocked with a single clock signal. An eight bit bus is used for data transfers to the processing elements which are addressed by the row and column lines. The same eight instructions of the previous design can be executed in the current design (refer to table 1).

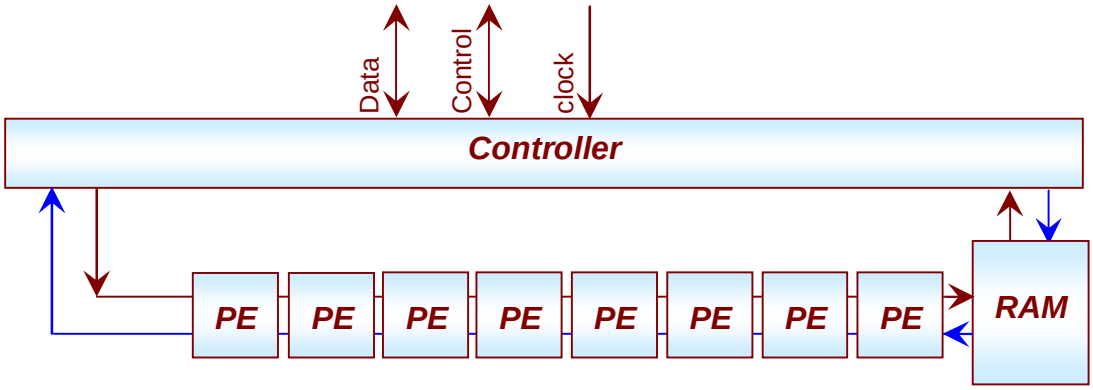


Figure 10: *8×8 Network Design Structure*

I- Processor Element Hardware Architecture

Each neuron (PE) consists of two blocks as shown in Figure 11. The weight memory block contains 8×16 byte of SRAM and is used to store the synapse weights for 8 neuron aligned in a certain one of eight rows while all the necessary operations for the data processing are implemented in the calculation unit block.

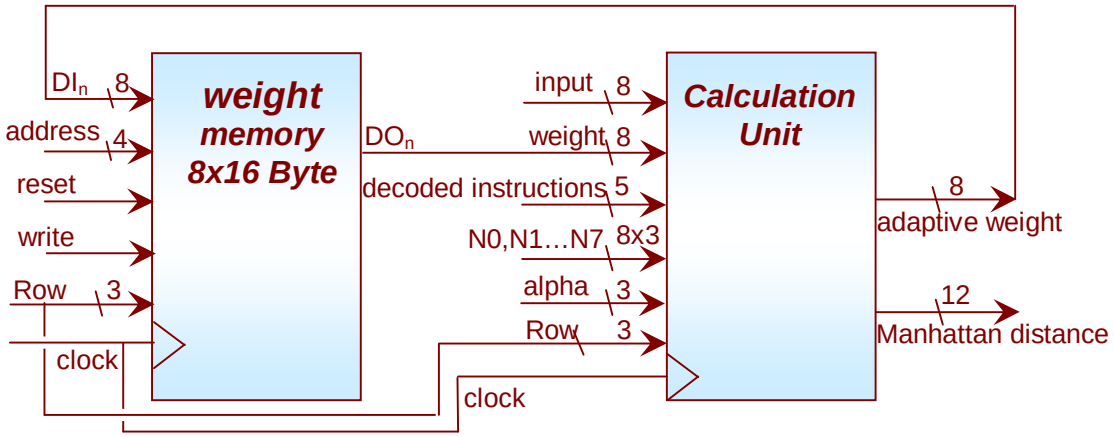


Figure 11: *One Processor Element Block Diagram*

A calculation unit usually does not become active until it gets a *Reset* on the instructions data line. The components of the input vector are transmitted over the input data lines and the distance to the weight vector component on the weight data line is calculated when the instruction on the instructions data line is *CalDist* (refer to table 2). This is done via eight stages. After all the distances have been calculated, the minimum one will be specified by the controller and adaptation starts. In order to realize the neighborhood function, a value from a lookup table (neighbor) is added to the value of input (alpha) when the instruction on the instructions data lines is *LoadAlpha*. Then the new weight vector components are calculated (one component per clock cycle). This is done when the instruction on the instructions data line is *UpdateWeight*. Some times there is a need for wait cycles between instructions. Here this is done by using the *NOP* instruction. Calculations are completed on eight stages. That is why it is called semi-parallel architecture.

Calculation Unit Hardware Architecture

The explained functions are performed by the following logical circuits of the calculation unit:

1. The circuit shown in Figure 5 is used to calculate the Manhattan distance. It is the same one used by the previous design (4×4Map).
2. The logic circuit shown in Figure 12 is used to calculate the amount of shifting operations used later in adaptation process. This circuit differs from the *LoadAlpha* instruction circuit in the 4×4 design in the input neighbors. Here there are eight neighbors controlled by the row lines.

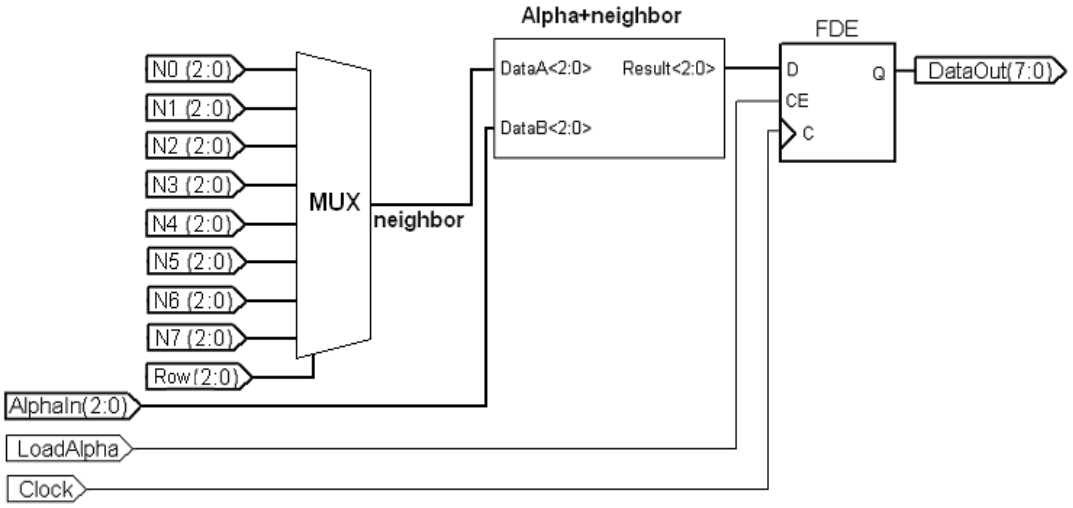


Figure12: *Load Alpha Instruction Circuit for 8×8 Map*

3. The logic circuit shown in Figure 7 is used to calculate the updated weight values during adaptation processing. The operation for this circuit is controlled by the *UpdateWeight* instruction.

II- Control Unit

The control unit performs many common functions like *Instruction decoding, Data Multiplexing,* Address generating using *Counter and Winner Estimator* for finding the best matching cell during learning. Figure 13 illustrates the *Winner Estimator* element. The rest of circuits are the same used in the previous design.

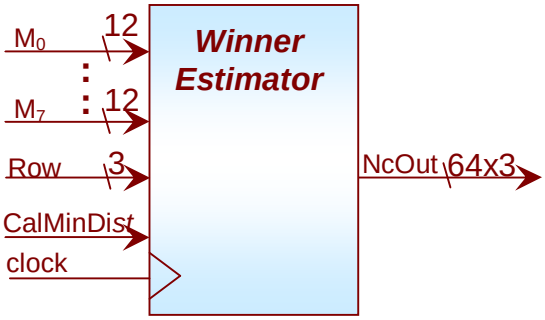


Figure 13: *The Winner Estimator Block Diagram*

4. Hardware Implementation and Results

Two types of SOM architectures are presented in this work, the full parallel 4×4 map and the semi-parallel 8×8 map. All SOMs are successfully passed the tests as the next results prove. The differences between the two designs are also presented with, their timing analysis and speedup.

In order to check the validity and the usability of the building Kohonen neural network hardware, function simulations have been done using MATLAB and ModelSimXE packages and the results have been examined before the designs are implemented on the real device. The original image (LENA image 512×512 pixels, 256 gray levels) used for the test is shown in figure 14(a), is tested on two-dimensional SOM designs in this work.

4.1 Full Parallel 4×4 Network Result

Figures 14(b) represents the decoded image produced using MATLAB while Figures 14(c) represents the decoded outcomes using ModelSimXE simulator which are comparable to the same results obtained from the chip under test. The SNR, PSNR, MSE, NMSE, CR_on, CR_off, bpp_on, bpp_off for two-dimensional SOM array with 4×4 size are summarized in table 3.

(a) Original Image *(b) MATLAB Results* *(c) Hardware Results*

Figure 14:*Original and Reconstructed Images Using SOM with 4x4 Neurons*

Table 3 :*Statistical Results for 4×4 SOM Map*

Image	SNR(dB)	PSNR(dB)	MSE	NMSE	CR_on	CR_off	bpp_on	bpp_off
MATLAB	20.0288	25.7493	173.04	0.98584	15.7538	16	0.5078	0.5
Hardware	15.8172	21.6739	442.277	2.5197				

Table (4) lists the chip area occupation for this design. This report is created during synthesize stages.

Each processing element can handle one vector component input per clock cycle. So, for an input vector of 16 components (8-bit accuracy), the calculation of the distance (12-bit accuracy) to the internal weight vector consumes 16 clock cycles. After that, the best match search requires 5 clocks by the controller. The adaptation factor α distribution requires 2 clock cycles. During learning, the weights of the best match element and the surrounding elements have to be adapted. The strength of the adaptation depends on the learning step and the position of the element corresponding to the best match position. The elements can adapt their weights in parallel during a time of 48 clock cycles {(read weight , update weight, write weight) *16 } plus 2 clock cycles for reset and NOP which leads to a total of 73 clock cycles required by the network, during learning, for one input vector, see Figures 15.

Table 4: *Device Utilization Summary for 4×4 SOM Map*

Device Type	Percentage area occupied
Number of Slices	1221 out of 1920 63%
Number of Slice Flip Flops	747 out of 3840 19%
Number of 4 input LUTs	2168 out of 3840 56%
Number of bonded IOBs	25 out of 173 14%
Number of BRAMs	9 out of 12 75%
Number of GCLKs	1 out of 8 12%
Minimum period: 11.244ns (Maximum Frequency: 88.936MHz)	

This means that the processing of each vector takes (73*1/50MHz)sec where (50 MHz) is the kit frequency. So a 1.46 microseconds is the time required for training by a single vector. The total training time for Lena which consists of 16384 vectors is 23.92064 milliseconds.

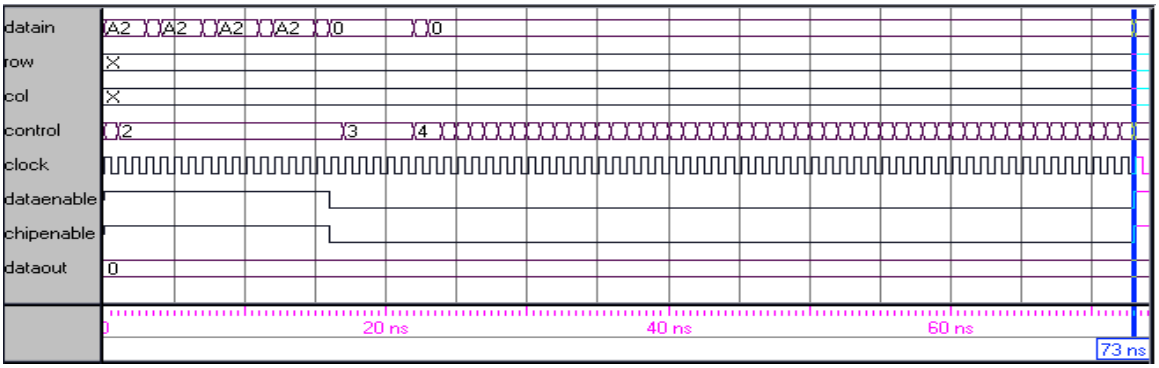


Figure 15: *Waveforms for One Input Vector During Learning with 4x4 SOM*

4.2 Semi-Parallel 8×8 Network Result

Figures 16(b) represents, approximately, the decoded images results using MATLAB while Figures 16(c) represent the decoded images results using ModelSimXE simulator which are the same results obtained from the chip under test. The SNR, PSNR, MSE, NMSE, CR_on, CR_off, bpp_on, bpp_off for two-dimensional SOM array with 8×8 size are summarized in table 5.

(a) Original Image *(b) MATLAB Results* *(c) Hardware Results*

Figure 16:*Original and Reconstructed Images Using SOM with 8x8 Neurons*

Table 5 :*Statistical Results for 8×8 SOM Map*

Image	SNR(dB)	PSNR(dB)	MSE	NMSE	CR_on	CR_off	bpp_on	bpp_off
MATLAB	20.4517	26.1710	157.0306	0.8946	15.0588	16	0.53125	0.5
Hardware	16.0689	21.4269	487.9631	2.78				

Table (6) lists the chip area occupation for this design. This report is created during synthesizes stage.

Table 6 : *Device Utilization Summary for 8×8 SOM Map*

Device Type	Percentage area occupied			
Number of Slices	273	out of	1920	14%
Number of Slice Flip Flops	230	out of	3840	5%
Number of 4 input LUTs	527	out of	3840	13%
Number of bonded IOBs	24	out of	173	13%
Number of BRAMs	4	out of	12	33%
Number of GCLKs	1	out of	8	12%
Minimum period: 9.130ns (Maximum Frequency: 109.529MHz)				

Each processing element can handle one vector component input per clock cycle. For an input vector of 16 components (8 bit accuracy), the calculation of the distance (12 bit accuracy) to the internal weight vector consumes 16 clock cycles. The distance calculation is overlapped with the best match search by the controller. The calculation of distances, with minimum estimation, takes 147 cycles. The adaptation factor α distribution overlapped with the adaptation for elements of the best matching cell with the surrounding ones during learning. This takes 393 clock cycle {load Alpha+ (read weight , update weight, write weight) *16 }*8 plus 1 this leads to 393 clock cycle. The overall cycles needed to learn for one vector are 540. So each vector takes (540*1/50MHz)sec where (50MHz) is the kit frequency, and (0.108) milliseconds is the time required for training by a single vector. The total training time for Lena image which consists of 16384 vectors is 1.769472 seconds. All signals can be shown in Figure 17.

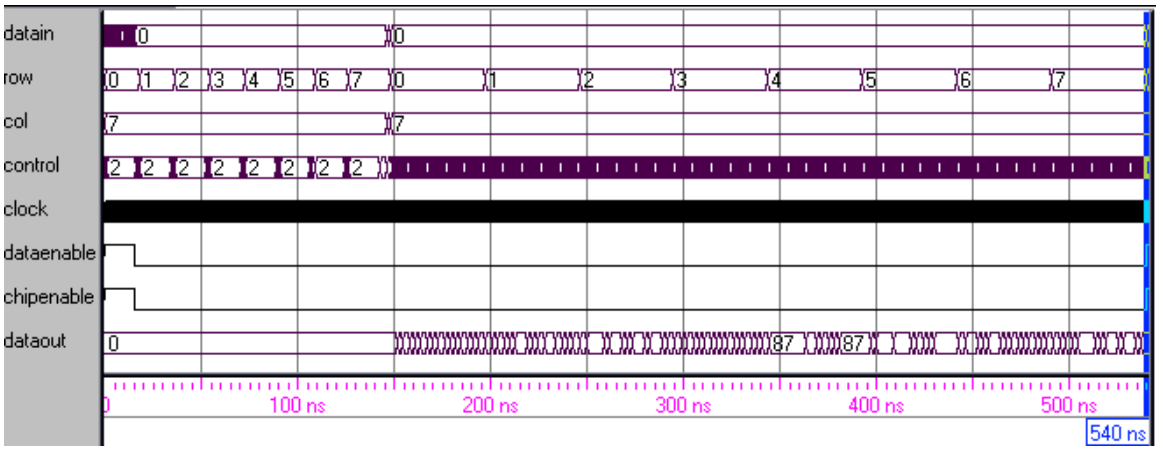


Figure 17: *Waveform for One Input Vector During Learning with 8x8 SOM*

5. Performance of the Systems

VHDL is used to describe a digital system at the behavioral level so that the designer can simulate the system to test or verify the algorithm used and to make sure that the sequences of operation are correct. This includes measurement of the performance, area occupied (a measure of the complexity of the design); and processing time (a measure of the running speed).

For the comparison of the design with other implementations of the self-organizing maps, the performance of the architecture is determined for the recall phase and for the learning phase. This can be done by using performance metrics CPS (Connections Per Second) for the recall phase and CUPS (Connection Updates Per Second) for the learning phase [8]. Furthermore, the classification rate and the adaptation rate are given (i.e. classified or adapted vectors per second respectively). All these values are summarized in table 7.

Table 7 : Hardware performance metrics

	4x4 Map	*8x8 Map*
CPS	12.8 GCPS	3.2 GCPS
CBS	51.2 GCBS	12.8 GCBS
CUPS	3.2 GCPS	800 MCPS
Classification rate	2380,000	340,000
Training rate	704,000	92,000

The implementation performance of the two designs are compared with each other (Figure 18). Also, the different speeds for each design are compared with that when softwarely running on computer (Pentium4, 2.26GHz, 248MB RAM). Refer to Figure19. Figure 20 illustrates the obtained speedup.

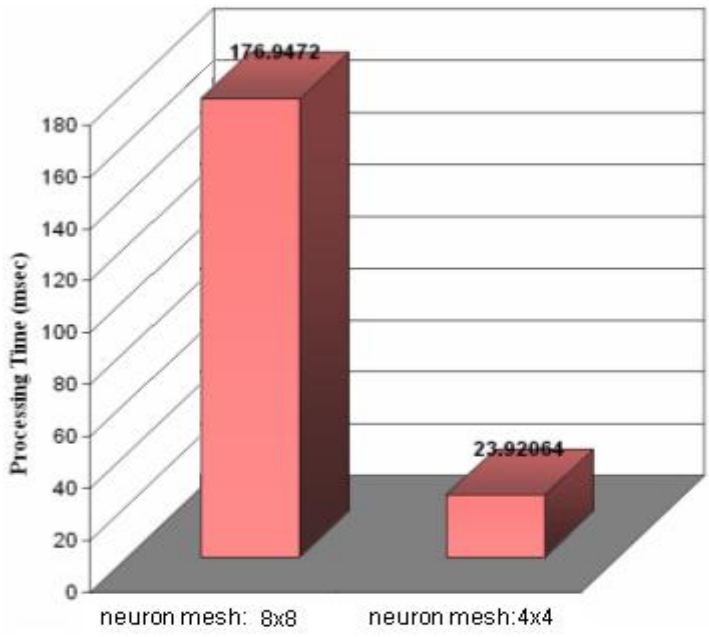


Figure18: *Processing Time of Parallel(FPGA) Processor*

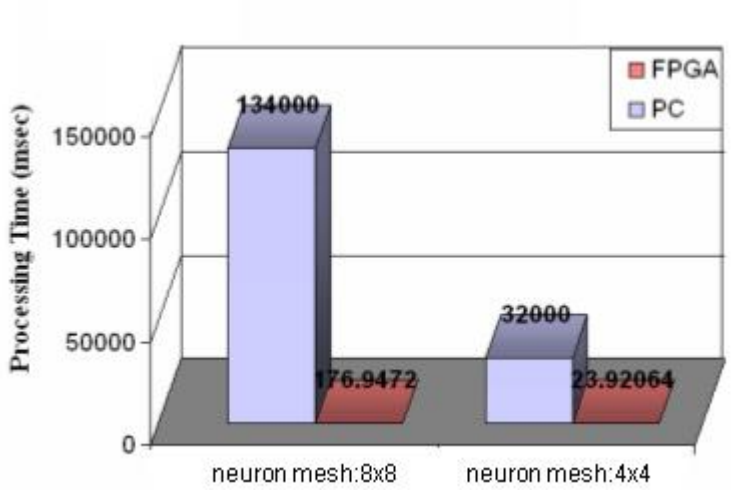


Figure19: *Processing Time of Parallel(FPGA) and Sequential(PC) Processors*

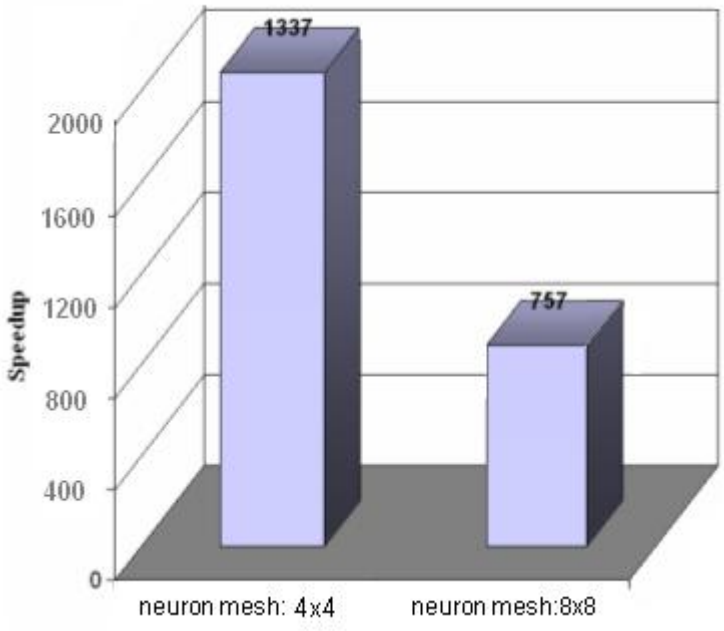


Figure 20: *Obtained Speedup for The Two Designs*

6. Conclusions

In this paper, we have presented for self organization map neural network two chips , designed , realized on FPGA and tested to carry out their designed functions. The results given verify that the Kohonen neural network chip is capable of on-chip learning where its processing elements have on-chip memory and therefore can perform on-chip learning. The first chip of 26 pin and 4x4 PEs arranged as a SIMD array can be easily extended to 8x8 or more and has a speed up of 1337 over the P4 computer of 2.26GHzfrequency . The second chip is of 28 pins and 8x8 PEs arranged also as a SIMD array but working in a semi-parallel , it has a speed up of 757 over the P4 computer of 2.26GHz frequency .

References

[1] F. Lara, *"Artificial Neural Networks: An Introduction*", Instrumentation and Development Vol. 3 Nr.9/1998.

[2] Misra, M., *"Parallel Environments for Implementing Neural Networks*", Neural Computing Surveys Vol. 1, 48-60, 1997.

[3] Yihua Liao,*" Neural Networks in Hardware: A Survey"*, Department of Computer Science, University of California, Davis One Shields Avenue, Davis, CA 95616. http://wwwcsif.cs.ucdavis.edu/~liaoy/research/NNhardware.pdf.

[4] J. Zhu and P. Sutton,*" FPGA Implementations of Neural Networks a Survey of a Decade of Progress"*. International Conference on Field Programmable Logic and Applications (FPL'03), pp. 1062-1066, LNCS 2778, Springer Verlag, Berlin, Heidelberg, 2003.

[5] Teuvo Kohonen, *"The Self-Organizing Map*", Proceedings of the IEEE, Vol.78, No. 9, September 1990.

[6] Xilinx, Inc., Datasheet *"Spartean-3 FPGA Family: Complete Data Sheet"*,DS099 January 17, 2005.

[7]Xilinx, Inc., *"Spartan-3 Starter Kit Board User Guide*",UG130 (v1.0) April 28, 2004.

[8]M. Porrmann, S. Rüping, U. Rückert *"SOM Hardware with Acceleration Module for Graphical Representation of the Learning Process"*, Proc. of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, MicroNeuro'99, Granada, April 7-9, 1999, pp. 380-386.

[9] Younis , B.,M.,K. ;"Hardware Design and Implementation of Kohonen Neurons ";Ph.D. Thesis ;University of Mosul ,College of Engineering; January 2007 .

The work was carried out at the college of Engg. University of Mosul