Journal of Advanced Computer Science and Technology, *2 (1) (2013) 28-37*
©Science Publishing Corporation
www.sciencepubco.com/index.php/JACST

Digital Hardware Pulse-Mode RBFNN with Hybrid On-chip Learning Algorithm Based Edge Detection

Amir Gargouri*, Dorra Sellami Masmoudi

Computer Imaging and Electronics Systems group from research unit on Intelligent Control, design & Optimization of complex Systems, University of Sfax, Tunisia
**Corresponding author E-mail: gargouriamir@yahoo.fr*

Abstract

A hardware implementation of pulse mode Radial Basis Function Neural Network (RBFNN) with on-chip learning ability is proposed in this paper. Pulse mode presents an emerging technology in digital implementation of neural networks thanks to its higher density of integration. However, hardware on-chip learning is a difficult issue, since the back-propagation algorithm is the most used, which requires a large number of logic gates in the hardware. To overcome this problem, we apply a hybrid process, which is split into two stages. In the first one, the K-means algorithm is used to update the centers of gaussian activation functions. Thereafter, the connection weights are adjusted using the back-propagation algorithm. Details of important aspects concerning the hardware implementation are given. As illustration of the efficiency and scalability of the proposed design, we consider edge detection operation which is a very important step in image processing. In the learning step, the RBFNN was taught the Canny operator behavior. Experiential results show good approximation features. The proposed design was implemented on a virtex II PRO FPGA platform and synthesis results showed higher performances when benchmarked against conventional techniques and neural ones.

Keywords: *Pulse mode, RBFNN, On-chip learning, K-means, Back-propagation, Edge detector.*

1 Introduction

Recently advances in artificial neural networks (ANNs) have led to several applications such as signal processing, face recognition, transfer function simulation and person biometric based identification. Radial basis function neural networks (RBFNN) constitute a particular class of neural networks widely used for regression and discrimination. Their theoretical and practical properties have been thoroughly studied since the late 80s. They are originally designed to implement some interpolation techniques of a set of points in a multidimensional space [1]. Moreover, RBF networks have the best approximation property, i.e. it can accurately approximate a nonlinear function [2], provided that the number of neurons is sufficient and the training algorithm is adequate. Furthermore, RBF network training is faster and easier thanks to hybrid learning algorithms, and requires fewer neurons compared to multilayer networks. In recent years, RBF networks have found a renewed interest in various applications notably in nonlinear function approximation, classification problems and image processing [3,4].
Although ANNs are usually implemented in software, many applications require implementation of fast and large neural networks on efficient custom device. Accordingly, VLSI (Very Large Scale Integration) implementation is crucial for building fast artificial neural networks, by incorporating various applications in artificial intelligence. Two main approaches to VLSI implementation of neural networks are applied. The first is the implementation that integrates in the same circuit the training and the generalization phases (on-chip learning); this hardware implementation is not easy, however, it is characterized by flexibility and adaptability to realise multiple applications. Whereas, the second implementation (off-chip learning) includes only the generalization phase, in such case, the network parameters are updated by training using software technology. Hardware implementation is accordingly devoted to the generalization phase using the network parameters set. However, the speed of software-based implementation is low, and it is not suitable for using in some environments where high performance is needed [5]. Size and real-time operations show that on-chip learning is necessary for a large range of applications.
In recent years, pulse mode architectures are gaining support in the field of hardware neural networks with on-chip learning ability, thinks to these outstanding features. In fact, pulse mode affords a good capacity to incorporate on-chip

applications in artificial intelligence and to reduce significantly hardware resources, by integrating a simple frequency multiplier [6-9]. In this paper, we propose a hybrid learning process for effective hardware implementation of RBFNN, consisting of the K-means algorithm to adjust the center positions of the gaussian functions, and the back propagation algorithm to estimate the connection weights. The proposed network can be used to approximate several image processing tasks. As an illustration example, we consider edge detection application.
The remainder of this paper is organized as follows, firstly in section 2, the RBF neural network characteristics are presented. Secondly, in section 3, we describe the pulse mode RBFNN with hybrid on-chip learning ability. In Section 4, we apply this network as illustration in edge detection. In Section5, simulation results of the whole network are presented and discussed. Finally, conclusions are drawn in Section 6.

2 Theoretical background

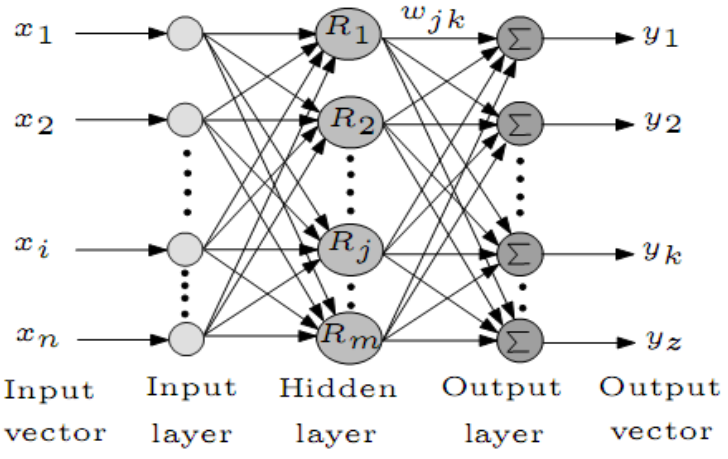
2.1 RBFNN architecture

The RBFNN architecture is shown in Fig.1. It involves one input layer, one hidden layer and one output layer. The neurons of the input layer are not connected to each other; meanwhile, they are directly connected to all the neurons of the hidden layer. The output neurons perform a linear combination of the nonlinear basic functions provided by the neurons of the hidden layer. There may be one or more output neurons, the number of hidden layer units is determined experimentally, because there is no analytical method giving its required number [10]. Thanks to these basic functions, RBFNN are able to provide a local spherical geometric representation of space and interpolate a set of points in a multidimensional space. Generally, the used activation functions are defined in the real interval [0,1], radially symmetrical around a center and characterized by an adjustable width. The response of the radial function decreases monotonically, following the removal of the input from its center [11].

The output is then taken as a linear combination of basis function outputs as follows.

$$y_k = \sum_{j=1}^{m} w_{jk} R_j(x) \tag{1}$$

Where W_{jk} (j=1...m and k=1...z) is the weight applied to the j^{th} hidden layer unit to get the k^{th} neuron of the output layer.



1

Fig.1: RBF neural network architecture

2.2. Training of RBFNN

Learning by training is probably the most important feature of neural networks, in which the network is updated to obtain the required behavior; such feature depends closely on the network architecture. In this work, for reducing the design consumed resources, the training procedure consists on a hybrid process: firstly, the K-means algorithm is implemented to update the network center values. Thereafter, the connection weights are adjusted using the back propagation algorithm.

3 Hardware implementation

In this section, the study will introduce the hardware implementation of the whole pulse mode RBFNN structure with hybrid on-chip learning ability. The overall component of the network is shown in Fig.2. The hardware implementation includes two parts: a feed forward path and a learning one.

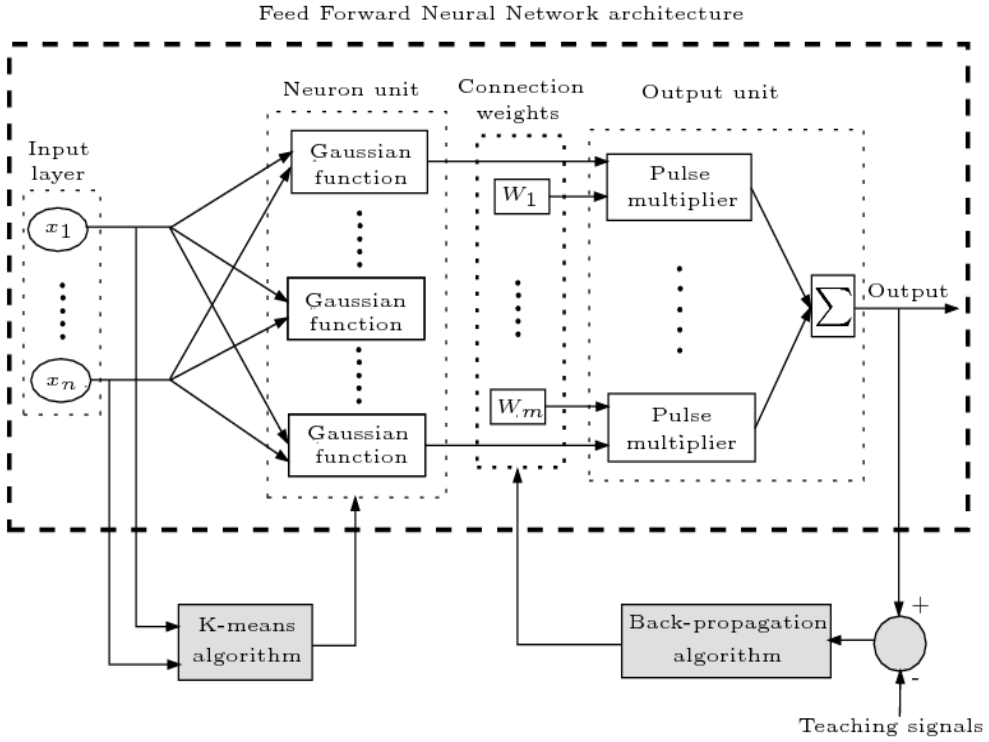


Fig.2: Pulse mode RBFNN architecture with hybrid on-chip learning.

3.1 Feed Forward Neural Network architecture

This work is drawn from a previous study of ours [12], where the feed forward RBFNN architecture was successfully implemented by integrating of three computational elements: a neuron unit which uses the gaussian function as activation function, a pulse multiplier which performs the weight multiplication and an output neuron which calculates the sum of synapse unit outputs. A more detailed discussion may be found in [12].

3.2 On-chip learning architecture

3.2.1 Implementation of the K-means algorithm

The K-means algorithm [13] is an unsupervised method, in which input data are divided into clusters. Then, each clock period, so-that each set of samples is updated to its nearest center, the number of centers which is equal to the number of hidden neurons is is empirically fixed. Thus, we undertake the following steps:

1. Randomly chose the initial centers of clusters among the data to classify. Let $c_1,...,c_j,...,c_n$ the different network centers
2. Each input is set to the cluster whose center is the nearest. Thus the distribution of inputs in each cluster is defined as follows:

   - The cluster that has a center of gravity c_1, contains all points equal or lower than $\frac{c_1+c_2}{2}$.

- The cluster that has a center of gravity c_j, integrates all points ranging in $]\frac{c_{j-1}+c_j}{2}, \frac{c_j+c_{j+1}}{2}]$.

- The cluster that has a center of gravity c_n, includes all points higher than $\frac{c_{n-1}+c_n}{2}$

3. At the end of each iteration, update the new center of each cluster according to equation (2), where X_j is the sum of the different samples belonging to the j^{th} cluster and N_j is the number of the samples.

$$c_j = \frac{1}{N_j} \sum_{n \in S_j} X_n \tag{2}$$

4. Repeat steps 2 and 3 until convergence is reached (updated centers of clusters do not change over iterations).

The proposed architecture of the K-means algorithm is depicted in Fig.3. After initialization of the different centers, we focus on the implementation of steps 2. Each clock sample, an input is introduced to the circuit to determine its cluster, thereafter, the content X_j of the j^{th} cluster is added with the input value, whereas the number N_j of samples is incremented. Subsequently, we proceed to step 3, which computes the new center of each cluster by applying equation (2). In order to enhance the hardware implementation, we select a power of two samples among N_j. This enables replacing dividers with shifters. Fig.4 describes the different waveforms in the K-means bloc. Its operation is as follows: if N_j is a power of two (let it be 2^{sj}), the content of the register R_{1j} inherited the value of N_j, while the content of the register R_{2j} takes the value of X_j. At the end of distribution of all inputs, we proceed to compute the new centers by dividing R_{2j} by R_{1j}, thus the divider is implemented using a right shift register.

An activation function is then defined by cluster, which corresponds to the center of gravity of each cluster. In this proposed work, to reduce complexity, the widths are determined so that we get a small overlap between the activation functions and do not proceed in their update. In the learning step, we obtained satisfactory results, as well as in the generalization test step.

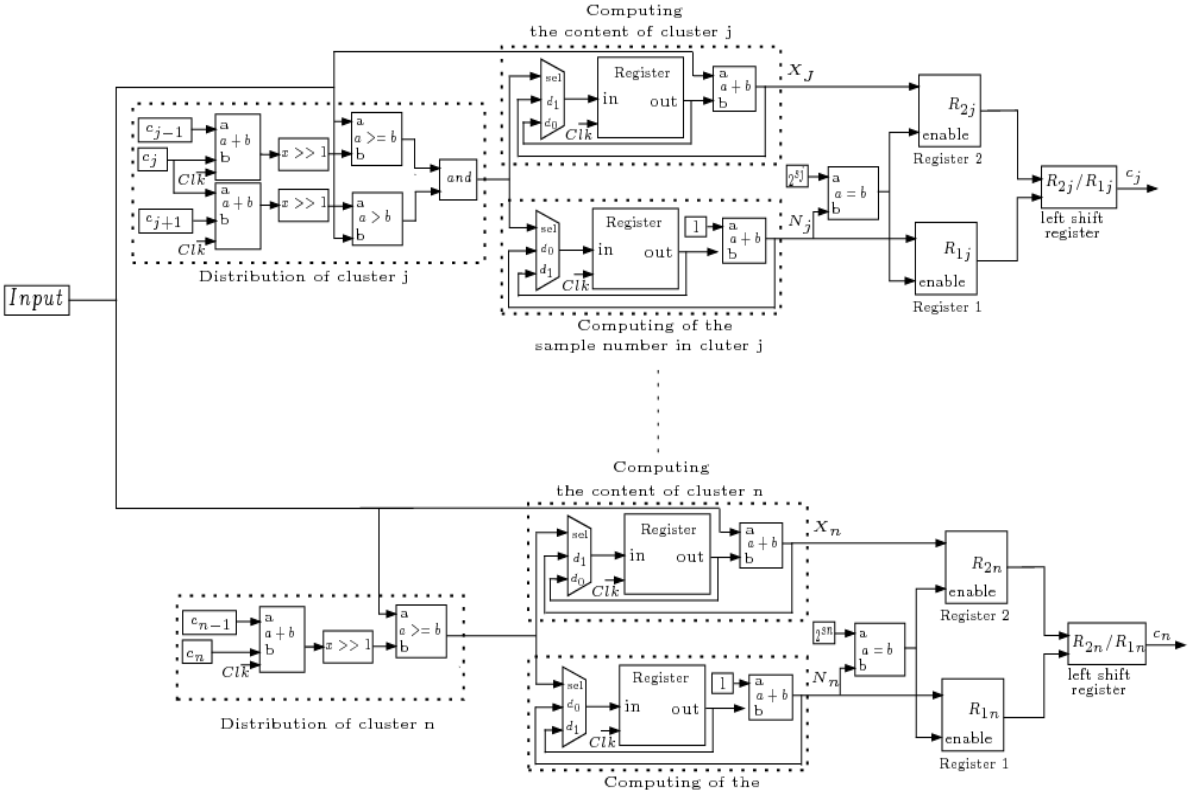


Fig.3: Architecture of the K-means algorithm.

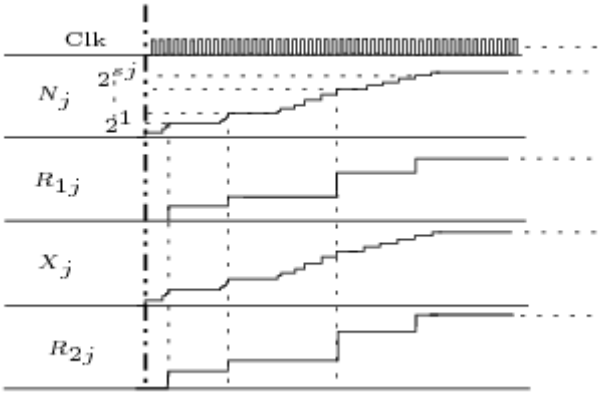


Fig.4: Different waveforms in the K-means bloc

3.2.2 Implementation of the back-propagation algorithm

The back propagation algorithm is a supervised method computed in the backward computation. Criterion for the learning algorithm is to minimize the error given by equation (3), where y_k is the output vector and y_{tk} is the output target.

$$E = \frac{1}{2}\sum_{k=1}^{m}(y_k - y_{tk})^2 \tag{3}$$

This error is closely related to the network parameters according to the following equations:

$$\Delta W_{jk} = -\rho_w \frac{\delta E}{\delta W_{jk}} = -\rho_w (y_k - y_{tk}) R_j(x) \tag{4}$$

$$\Delta c_j = -\rho_c \frac{\delta E}{\delta c_j} = -\rho_c \frac{x_j - c_j}{\sigma_j^2} R_j(x) \sum_{k=1}^{m} (y_k - y_{tk}) W_{jk} \tag{5}$$

$$\Delta \sigma_j = -\rho_\sigma \frac{\delta E}{\delta \sigma_j} = -\rho_\sigma \frac{(x_j - c_j)^2}{\sigma_j^3} R_j(x) \sum_{k=1}^{m} (y_k - y_{tk}) W_{jk} \tag{6}$$

Finally, an update all RBF network parameters using the equations below:

$$\Delta W_{jk} = W_{jk}(t+1) - W_{jk}(t) \tag{7}$$

$$\Delta c_j = c_j(t+1) - c_j(t) \tag{8}$$

$$\Delta \sigma_j = \sigma_j(t+1) - \sigma_j(t) \tag{9}$$

The optimization problem is nonlinear and iterative; the stopping criterion can be a maximum number of iterations or a predefined minimum error. In our case, we apply the first solution.

For the implementation, we deduce from equation (7), a whole scheme making use of different basic blocks depicted in Fig.5. The learning process begins by calculating the error once the network output is generated. Then, it is multiplied by ρ_w using a constant multiplier which is implemented using a left shift register. Afterwards, the quantity $\rho_w \times (y_k - y_{tk})$ is multiplied by the hidden neuron output R_j using a pulse multiplier, this multiplication is performed during a period T_f. Finally, we define an enable signal T_w, and at its rising edge we actualize the weight values to be used in the next iteration. Fig.6 shows the different waveforms in the weight update block.

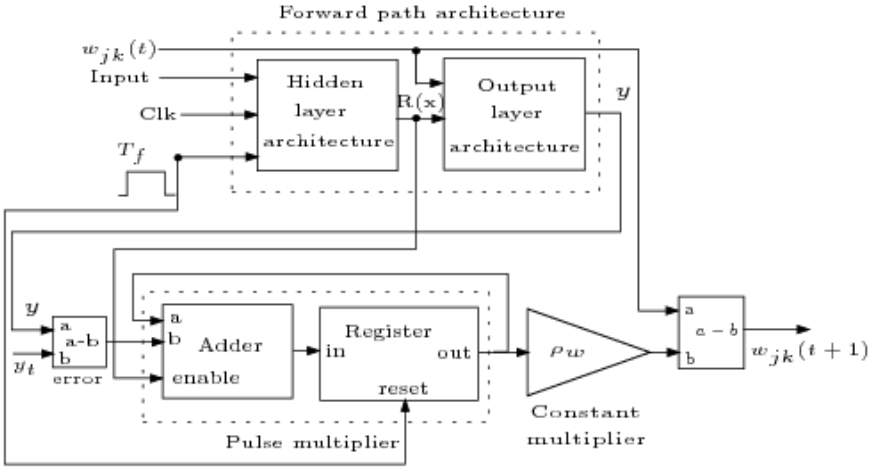


Fig.5: Weight update block

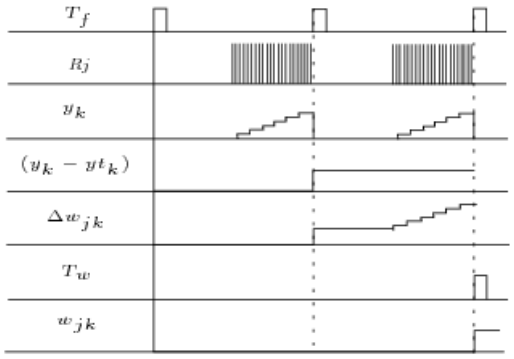


Fig.6: Waveforms in the weight update block.

4 Pulse mode RBFNN based edge detection

Genericity and updating of different parameters in neural networks can be seen as an easy bloc reconfiguration. Such ability affords a great potential to reduce a system hardware resources cost by configuration of the network and making it achieving different processing tasks. In this work, the proposed pulse mode RBFNN is used as a prevalent technology, instead of classic edge detection methods. Edge detection presents a fundamental step in image processing. This stems from the fact that edges characterize the boundaries of the image objects, and are therefore, the base for subsequent applications such as image segmentation, boundary detection, object recognition and classification, image registration, and so on. Consequently, the successes of these subsequent image processing tasks are strictly dependent on the performance of edge detection. In this paper, we consider the Canny edge detection operator for modeling and implementing the RBFNN. The Canny detector which is one of the most known border estimators in the literature and it is widely used in several applications [14-15]. The Canny edge detector algorithm is based on several steps. Firstly, it removes noise by a smoothing process. Secondly, it computes the image gradient to deduce regions with high derivatives. Then, the algorithm tracks along these regions and suppresses any pixel that is not at the maximum [16]. In this work, we try to include all these complex steps into a neural network based approach.

4.1 Learning step

The internal parameters of the proposed neural edge detector are optimized by using hybrid learning algorithm. Fig.7 represents the setup used for training. Firstly, the center positions of the gaussian functions are adjusted using the K-means algorithm. Afterwards, the connection weights are updated by means of the back-propagation algorithm. In our design, we constituted our database with different images from Wangz image database [17] such as dinosaur images,

fusil images, etc... Each set is divided on two classes, one is for the network training, and the other is for its generalization test. For the back-propagation algorithm, we set to the network the input images and their corresponding of the canny edge detector outputs as the target ones. For the K-means algorithm, we used only the input images. The structure of the used network consists of nine inputs, four hidden neurons and one output neuron.

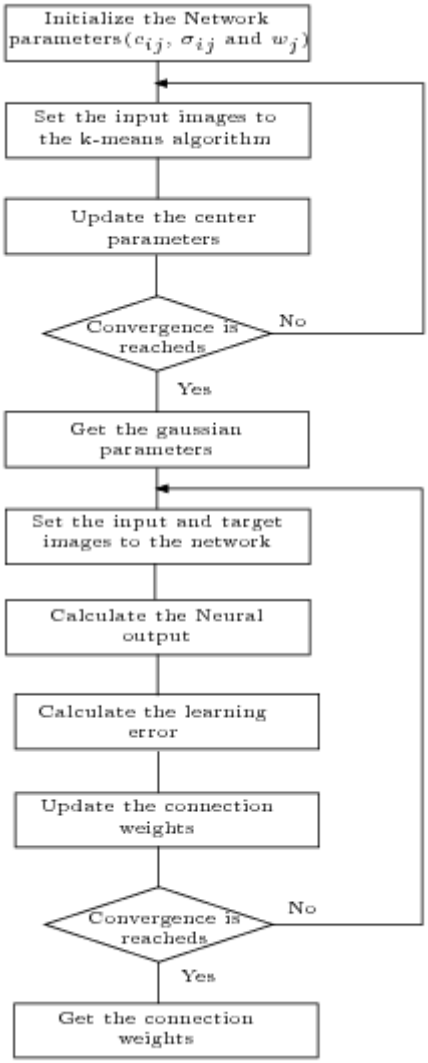


Fig.7: Hybrid on-chip learning steps for edge detection.

4.2 Experimental results

Performance of the proposed neural edge detector and its generalization abilities are tested by implementing the designed architecture in the FPGA platform. Different test images from the nontraining database were used. All test images are 8-bit grey level images having the same size. Fig.8 shows generalization results of the network.

To be able to evaluate the efficiency of the proposed edge detector, a metric based on the normalized mean square error (NMSE) is used, it is given by:

$$NMSE = \frac{\sum_{r=1}^{R}\sum_{c=1}^{C}[x(r,c)-y(r,c)]^2}{\sum_{r=1}^{R}\sum_{c=1}^{C}[x(r,c)]^2} \quad (10)$$

Where x is the image resulting from network output and y is the image processed by Canny operator. Less NMSE values show good edge detection results.

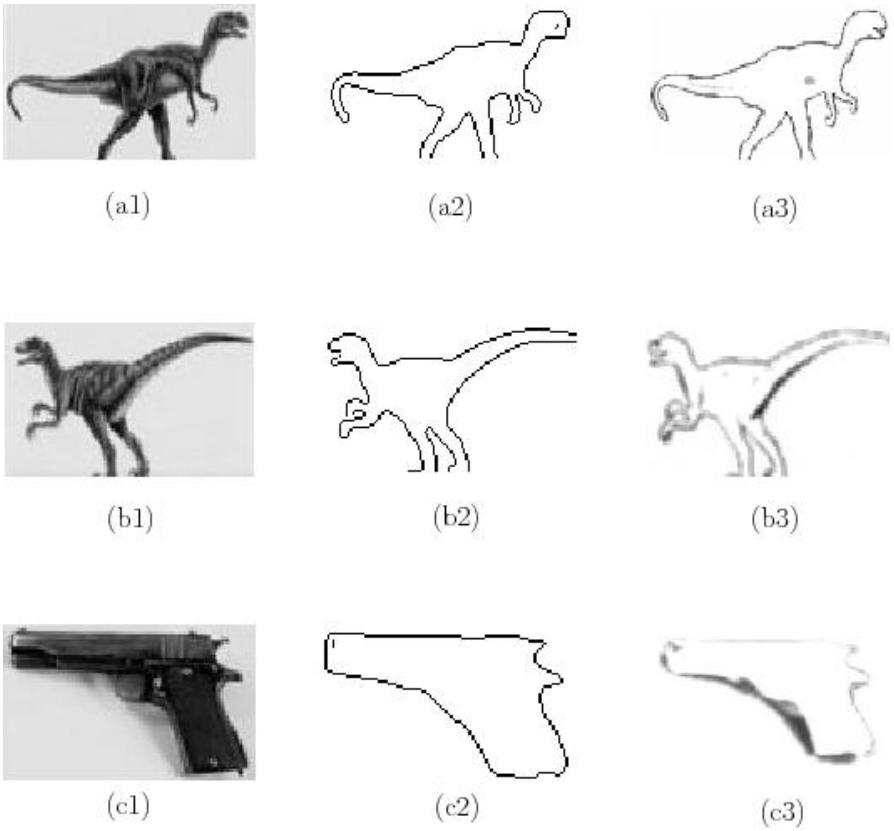


Fig.8: Generalization test results for the neural network based edge detection: (a1),(b1),(c1): Input images, (a2),(b2),(c2): Target images, (a3),(b3),(c3): Output images

Table 1 presents relative errors with respect to the network generalization. It is clear that the designed network is able to perform the edge detection operations; such results prove the efficient hybrid on-chip learning and the good generalization abilities.

Table 1: NMSE values calculated for the RBFNN based edge detection.

Images	NMSE values
(a3)	3.8%
(b3)	4.1%
(c3)	4.8%

5 Simulation results and synthesis

In this section, we implement the whole architecture of the pulse mode RBFNN with hybrid on-chip learning ability. The implementation is realized on a (Virtex 2p7 Fg456) FPGA chip. All the connection weight parameters w_{jk} are expressed in 16-bits fixed-point. The period T_f is set to 256 clock period to encode the unsigned inputs (pixels) presented in 8-bit. In the first one, we implement the K-means algorithm, its resources requirement is shown in Table 2. Afterwards, the back-propagation is implemented; Table 3 illustrates its device utilization. It can be seen that the FPGA chip is sufficient to perform the whole pulse mode RBFNN architecture, thus a larger network can be realized without sacrificing its response time, which enable to perform real time applications. Comparisons with conventional techniques and Neural ones [5,12], reveals the compactness of the suggested hybrid on-chip learning design and its ability to reduce the hardware resource costs.

Table 2: Device utilization and timing summary of the K-means algorithm.

Features	Utilized
Number of external IOBs	113 out of 396 : 28%
Number of slices	727 out of 4928 : 14%
RAM blocks	4 out of 44 : 9%
Clock blocks	1 out of 16 : 6%
Minimum period of the clock	9.826 ns
Maximum frequency	101.771 MHz

Table 3: Device utilization and timing summary of pulse mode RBFNN with on chip back-propagation algorithm.

Features	Utilized
Number of external IOBs	113 out of 396 : 28%
Number of slices	3467 out of 4928 : 70 %
RAM blocks	36 out of 44 : 81%
Clock blocks	1 out of 16 : 6%
Minimum period of the clock	19.292 ns
Maximum frequency	51.835 MHz

6 Conclusion

In this paper, a pulse mode RBFFNN with hybrid on-chip learning has been discussed. The complexity of the on-chip learning design and the difficulty of the hardware implementation are greatly reduced using a hybrid learning process, consisting of the K-means algorithm and the back-propagation one. The most important feature of the proposed design is the simplicity of implementation and the easy of programmability, not making use of any conventional multiplier, which reduce significantly the learning time and enhance the hardware implementation.

Performance of the proposed design and its generalization abilities are validated by approximation of edge detection function. Experimental results have been shown that the network can approximate well edge detection operation and allows good generalization rates, in term of the normalized mean square error (NMSE). The whole network with on-chip learning is implemented on FPGA platform. Such implementation offers many advantages over other solution with respect to both hardware implementation cost and device timing performance.

References

[1] J.K. Sing, D.K. Basu, N. Mita, and M.K. Nasipuri, “Face recognition using point symmetry distance-based rbf network,” Applied Soft Computing, vol.7,(2005), pp58-70.

[2] W. Guo, T. Qiu, H. Tang, and W. Zhang, “Performance of RBF neural networks for array processing in impulsive noise environment,” Digital Signal Processing, vol. 8, (2008),pp168-178.

[3] D. Du, K. Li, M. Fei, “A fast multi-output rbf neural network construction method,” Neurocomputing, (2010).

[4] S. Song, Z. Yu, X. Chen, “A novel radial basis function Neural network for approximation,” International Journal of Information Technology, vol.11, no.9, (2005).

[5] O. Polat, T. Yildirim, “FPGA implementation of a General Regression Neural Network: An embedded pattern classification system,” Digital Signal Processing, vol. 20, (2010),pp.881–886.

[6] D.E. Van Den Bout, T.K. Miller III “A Digital Architecture Employing Stochasticism for the Simulation of Hopfield Neural Nets,” IEEE Trans. Circuits Systems, vol. 36, n.5(1989).

[7] M. Krid, A. Dammak, and D.S. Masmoudi, “Hardware implementation of pulse mode RBF neural network-based image denoising,” Int. J. Electron. Commun. (AEU), vol. 63, (2009),pp.810–820.

[8] H. Hikawa, “Frequency-Based Multilayer Neural Network with On-Chip Learning and Enhanced Neuron Characterisitcs,” IEEE Trans. Neural Networks, vol. 10, no. 3, (2003).

[9] Y. Maeda, and T. Tada, “FPGA Implementation of a Pulse Density Neural Network With Learning Ability Using Simultaneous Perturbation,” IEEE Trans. Neural Networks, vol.14, no. 3, (2003).

[10] Kaliraj & Baskar, “An efficient approach for the removal of impulse noise from the corrupted image using neural network based impulse detector,” Image and Vision Computing, vol. 28, (2003), pp.458–466.

[11] D. Casasent, and X. Chen, “New training strategies for RBF neural networks for X-ray agricultural product inspection,” Pattern Recognition, vol. 36,(2003), pp.535- 547.

[12] M. Krid, A. Gargouri, and D. S. Masmoudi, “Hardware implementation of pulse mode RBF neural network-based image denoising,” Int. J. Innovative Computing and Applications, vol. 3, no. 4, (2011).
[13] Y. Zhao, F. E.H. Tay, F. Siong Chau, and G. Zhou, “Linearization of the scanning field for 2D torsional micromirror by RBF neural network,” Sensors and Actuators, vol. 121, pp.230–236, (2005).
[14] A. Basturk, and E. Gunay, “Efficient edge detection in digital images using a cellular neural network optimized by differential evolution algorithm,” Expert Systems with Applications, vol. 36, (2009),pp.2645–2650.
[15] M.E. Yuksel, “Edge detection in noisy images by neuro-fuzzy processing,” Int. J. Electron. Commun. (AEU), vol. 61, (2007),pp.8 –89.
[16] L.M. Reyneri, “Edge A performance analysis of pulse stream neural and fuzzy computing system,” IEEE Trans Circuits Syst, vol. 42,(1995) pp.624–60.
[17] J. Z. Wang, z. James, "wang research group: Databases for research comparison,” [Online]. Available: http:// wang.ist.psu.edu/docs/ related. shtml