Sensors **2014**, *14*, 18223-18243; doi:10.3390/s141018223

OPEN ACCESS
sensors
ISSN 1424-8220
www.mdpi.com/journal/sensors

Article

Parallel Fixed Point Implementation of a Radial Basis Function Network in an FPGA

Alisson C. D. de Souza and Marcelo A. C. Fernandes *

Department of Computer Engineering and Automation, Center of Technology, Federal University of Rio Grande do Norte—UFRN, Natal 59078-970, Brazil; E-Mail: alisson.camara@gmail.com

* Author to whom correspondence should be addressed; E-Mail: mfernandes@dca.ufrn.br; Tel.: +55-84-3215-3771; Fax:+55-84-3215-3738.

External Editor: Vittorio M.N. Passaro

Received: 6 July 2014; in revised form: 16 September 2014 / Accepted: 16 September 2014 / Published: 29 September 2014

Abstract: This paper proposes a parallel fixed point radial basis function (RBF) artificial neural network (ANN), implemented in a field programmable gate array (FPGA) trained online with a least mean square (LMS) algorithm. The processing time and occupied area were analyzed for various fixed point formats. The problems of precision of the ANN response for nonlinear classification using the XOR gate and interpolation using the sine function were also analyzed in a hardware implementation. The entire project was developed using the System Generator platform (Xilinx), with a Virtex-6 xc6vcx240t-1ff1156 as the target FPGA.

Keywords: artificial neural network; ANN; radial basis function; RBF; FPGA; fixed point; Simulink; system generator

---

1. Introduction

Artificial neural networks (ANNs) are computational techniques that employ mathematical models inspired by the neural structures of intelligent organisms, which acquire knowledge based on past and present experience. These intelligent organisms possess extremely complex sets of cells, called neurons; the structure of an ANN is composed of processing units called artificial neurons, whose functioning involves parallel and distributed interconnections [1]. One of the popular ANN architectures is the radial basis function (RBF) networks that employ least mean square (LMS) algorithms in their training.

From the implementation perspective, one of the main problems of RBFs relates to the lack of a methodology for the definition of their topology in terms of the number of centers, which, on the one hand, has a positive influence on their computational cost. There are several heuristic techniques available to assist the designer with network topology [1]. However, there is a consensus that the topology depends on the problem under investigation [2]. A commonly used procedure is to test various topologies for different sets of information. Nonetheless, this practice is time-consuming and can only be applied in cases of off-line training or when statistical information concerning the data space is known.

Various solutions for the implementation of application-specific integrated circuits (ASICs) have been proposed in order to accelerate the functioning and training of ANNs [3–5]. However, implementations in ASICs fix the architecture and the algorithm implemented, resulting in poor flexibility and/or high cost. Nevertheless, with advances in reconfigurable hardware structures, there has been renewed focus on the implementation of ANNs, and dedicated hardware structures are available that are flexible in terms of their topology and training algorithm. Currently, the most widely used architectures for reconfigurable hardware are the FPGAs, which can provide performance similar to an ASIC, with the advantage of rapid prototyping. There have been several studies concerning the implementation of multilayer perceptrons (MLPs) in FPGAs [2,3,6–11], the implementation of RBFs in FPGAs [12–21], as well as the implementation of SOMin FPGA [22].

This work presents a parallel hardware implementation of an RBF-type ANN. The implementation is made at a fixed point and is aimed at reconfigurable hardware architectures of the type FPGA. Different from earlier studies [2,3,6–8,12–20], here, a detailed analysis of the implementation of the RBF is provided, considering aspects, including processing time, delay time and the precision of the response. The proposed method was tested using two scenarios that have been widely reported in the literature. The first concerns the problem of nonlinear classification associated with the XOR gate, and the second considers the problem of interpolation using the sine function. All of the results were obtained using the System Generator (Xilinx) development platform [23], with a Virtex-6 xc6vcx240t-1ff1156 FPGA. The System Generator is a design tool over Simulink of MATLAB [24].

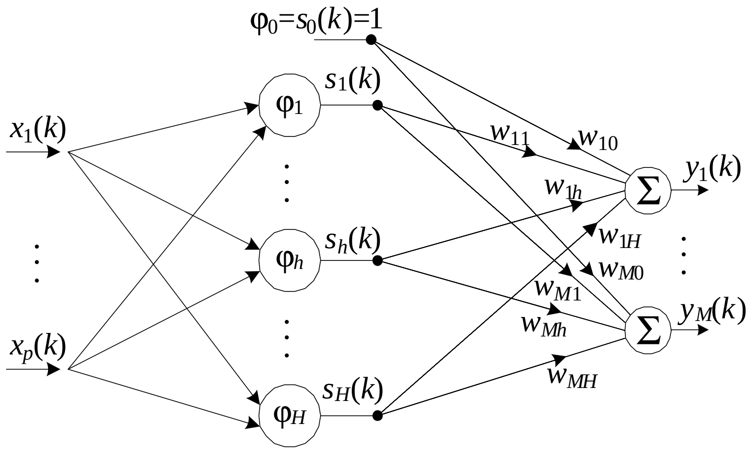
2. Radial Basis Function Networks

RBF networks have become increasingly popular in the domain of artificial networks [1]. The structure of an RBF network consists of multiple layers; the processing is the feedforward type, and the training can be either supervised (as used in the present work), or hybrid, where a supervised method is combined with an unsupervised method.

2.1. Architecture

The basic structure of an RBF network (Figure 1) consists of only three layers. The first layer is the connection of the model with the medium and is composed of p inputs. The second (or hidden) layer is composed of H radial basis functions (also known as neurons) and performs a nonlinear transformation of the input vector space into an internal vector space, whose dimensions are usually larger ($H > p$). The final (output) layer transforms the internal vector space into an output using a set of M linear neurons.

Figure 1. Architecture of the radial basis function (RBF).



Radial basis functions only produce responses that are significantly different from zero when the input pattern is located within a small region of the domain, and each function requires a center and a scaling parameter, β. The most widely used radial basis function is the Gaussian function [1]. The h-th function, $\varphi_h(\cdot)$, can be expressed by:

$$s_h(k) = \varphi_h(\mathbf{x}(k)) = e^{(-\beta v_h(k))} \tag{1}$$

where $\beta = \frac{1}{2\sigma^2}$ and $v_h(k)$, which represents the input distance, $\mathbf{x}(k)$, in relation to the h-th center, $\mathbf{c}_h$, expressed as:

$$v_h(k) = \|\mathbf{x}(k) - \mathbf{c}_h\|^2 \tag{2}$$

The vector of centers associated with the h-th radial function is characterized as:

$$\mathbf{c}_h = \begin{bmatrix} c_{h,1} \\ \vdots \\ c_{h,i} \\ \vdots \\ c_{h,p} \end{bmatrix} \tag{3}$$

and the vector of inputs as:

$$\mathbf{x}(k) = \begin{bmatrix} x_1(k) \\ \vdots \\ x_i(k) \\ \vdots \\ x_p(k) \end{bmatrix} \tag{4}$$

Equation (2) can therefore be rewritten and expressed by:

$$v_h(k) = \sum_{i=1}^{p} (x_i(k) - c_{h,i})^2 \tag{5}$$

The output of the m-th neuron of the output layer, N_m, can be characterized as:

$$y_m(k) = \sum_{h=0}^{H} w_{mh} s_h(k), \text{ for } m = 1, \ldots, M \tag{6}$$

Substituting Equation (1) in Equation (6) gives:

$$y_m(k) = \sum_{h=0}^{H} w_{mh}(k) e^{\left(-\frac{1}{2\sigma_h^2}\|\mathbf{x}(k)-\mathbf{c}_h\|^2\right)} \tag{7}$$

where w_{mh} is the synaptic weight between neuron h of the hidden layer and neuron m of the output layer. At each instant k, the RBF network receives a vector of inputs, $\mathbf{x}(k)$, and generates an output vector, $\mathbf{y}(k)$, expressed by:

$$\mathbf{y}(k) = \begin{bmatrix} y_1(k) \\ \vdots \\ y_m(k) \\ \vdots \\ y_M(k) \end{bmatrix} \tag{8}$$

2.2. Training Algorithm

Due to the difference between the hidden layer and the output layer, the training of RBF networks is usually divided into two parts. The first part concerns the nonlinear optimization performed by the hidden layer. In this, the input vector, $\mathbf{x}(k)$, is processed by means of functions present in the hidden layer, characterized by Equation (1). Several different strategies can be used to determine the centers, $\mathbf{c}_h$ [1]. Different from conventional methods, where fixed centers are selected randomly [1], the strategy employed here was to select fixed centers deterministically. The second part of the training involved calculation of the weights, w_{mh}, between the hidden layer and the output layer. The weights can be obtained using the pseudo-inverse method [1] or the LMS algorithm [25]. The online LMS procedure used here is an iterative technique that optimizes the mean squared error (MSE) function using the gradient descent method with classical stochastic estimation (exchanging the mathematical approach in favor of an instantaneous estimate) [25]. The parameters are adjusted every instant, k, using the expression:

$$w_{mh}(k) = w_{mh}(k-1) + \mu e_m(k) s_h(k) \tag{9}$$

where μ is the learning rate and $e_m(k)$ is the training error associated with the m-th output neuron in the k-th instant. The error can be expressed by:

$$e_k(k) = d_m(k) - y_m(k) \tag{10}$$

where $d_m(k)$ is the desired value for the m-th output neuron. The online LMS is less computationally complex, compared to the pseudo-inverse method that requires a nonquadratic matrix inversion calculation in order to obtain the weights, $w_{mh}(k)$ [25].

3. Architecture and Implementation

3.1. General Structure

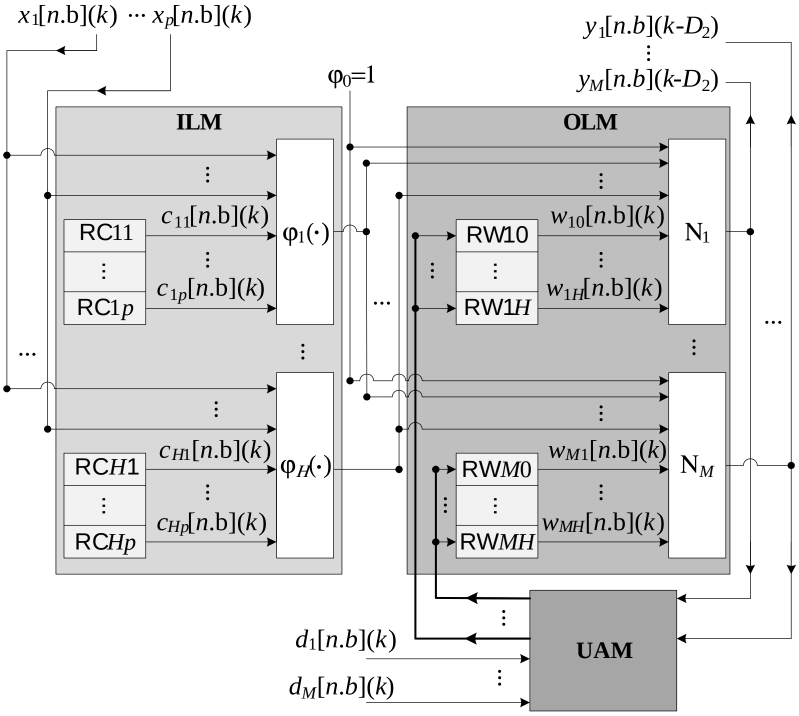
Figure 2 presents the general architecture of the proposed implementation. All of the variables and constants are implemented in a fixed point, utilizing a resolution of n bits, of which b bits represent the fractional part and $(n - b)$ bits represent the integer part. The representation $[n.b]$ is used for the fixed point variables with a sign, and $[\mathrm{U}n.b]$ is used for the variables without a sign. The architecture is composed of three large modules, characterized as the intermediate layer, output layer and updating algorithm, as illustrated in Figure 3.

(1) Intermediate layer module (ILM): This consists of the calculation of the radial basis functions, according to Equation (1), to generate the signal, $s_h(k)$, from the input, $\mathbf{x}(k)$. Processing of the H radial functions, $\varphi_h(\cdot)$, is performed in parallel, from the centers stored in the local registers possessing n bits. Each center, $c_{h,i}$, (see Equation (3)) is stored in the register RChi, as illustrated in Figure 2.

(2) Output layer module (OLM): This represents the processing performed by the M output neurons, as described in Equation (6). In this module, the M neurons also function in parallel, generating the signal $\mathbf{y}(k)$ (see Equation (8)) from the input, $\mathbf{x}(k)$.

(3) Updating algorithm module (UAM): This module is responsible for implementing the updating algorithm, which, in the present case, is the online LMS. The updating of each synaptic weight, $w_{mh}(k)$, at the k-th instant, is performed in parallel, according to Equation (9). As shown in Figure 2, the weights, w_{mh}, are stored in local registers of n bits, here termed RWmh.

Figure 2. General structure of the RBF network implemented in an FPGA.



The ILM, OLM and UAM modules function sequentially. For each instant k, the vector of inputs, $\mathbf{x}(k)$, is processed to generate the output vector, $\mathbf{y}(k)$, and all of the synaptic weights are calculated in order to update the registers, RWmh. It is important to note that it would be possible to further improve the performance of the RBF by using the modules in the form of a pipeline (this was not employed in the present work).

3.2. Radial Basis Functions

Figures 3 and 4 present details of the processing associated with the h-th radial basis function of the ILM.

Figure 3 illustrates the implementation corresponding to Equation (5). In this case, the overall implementation is performed in a partially parallel manner, as described previously [2,7,26]. The delays associated with the additions and multiplications are also shown in Figure 3. Each addition can be implemented with a delay z^{-a} and each multiplication with a delay z^{-r}. The insertion of delays in the operations relaxes the conditions of routing between the cells in the FPGA, principally in complex operations, such as those involving multipliers. On the other hand, the introduction of delays slows the response of the system, which can often be disadvantageous [23]. The calculation of the total delay, D, with respect to Equation (5), can be expressed as:

$$D = r + a + a\log_2(p) \tag{11}$$

Figure 3. The implementation as described in Equations (2) and (5).

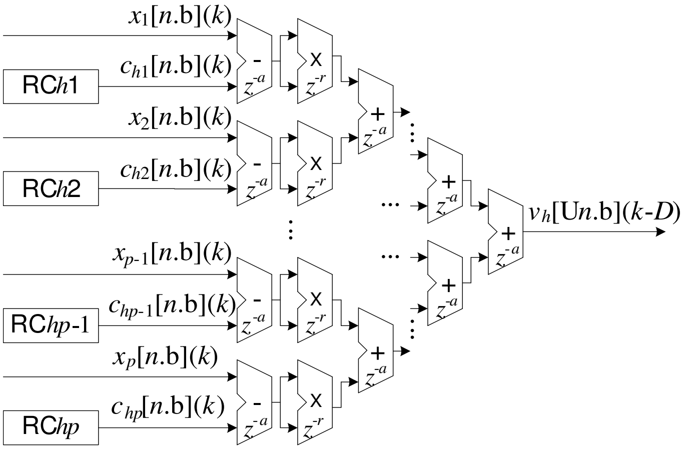
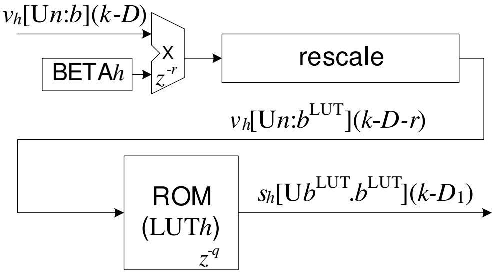


Figure 4. The implementation with respect to Equation (1).



A variety of techniques are available for the calculation of nonlinear functions in hardware, and in the case of FPGAs, one of the most common is the use of lookup tables (LUTs) in ROM memory [2,7,26]. Figure 4 shows the proposed implementation for the h-th radial function (see Equation (1)), in which read-only memory (ROM) was used to develop the LUT. For each h-th radial function, there is an LUTh and a multiplication operation to adjust the scaling parameter, β (stored in the BETAh register). In order to obtain greater resolution associated with each h-th LUTh, the $v_h(k)$ variable can be rescaled to a new format expressed by $\left[\mathrm{U}n.b_h^{LUT}\right]$, in which b_h^{LUT} is a new value for the number of bits of the fractional part, calculated using:

$$b_h^{LUT} = n - \left\lceil \log_2 \left(\left\lceil \delta_h^{max} \right\rceil \right) \right\rceil \tag{12}$$

where:

$$\delta_h^{max} = \max \left\{ v_h \left[n.b\right] \cdot \beta \right\} \tag{13}$$

Equation (13) determines the greatest distance of all of the input sets to the h-th center, and this information enables the number of bits of the integer part to be reduced to ($n - b_h^{LUT}$), while increasing the number of bits of the fractional part in b_h^{LUT}. Figure 4 illustrates the re-scaling step performed after the multiplication operation and before the LUT step.

As $v_h(k) \cdot \beta > 0$, for any k, the value of the response of the radial function is limited to $0 \leq s_h(k) \leq 1$ and can therefore be represented at a fixed point as $\left[\mathrm{U}b_h^{LUT}.b_h^{LUT}\right]$, in which the number of bits of the integer part is zero. The values stored in the h-th LUT can be characterized by the vector LUT$_h$, expressed as:

$$\mathrm{LUT}_h = \begin{bmatrix} s_h^0 \left[\mathrm{U}b_h^{LUT}.b_h^{LUT}\right] \\ \vdots \\ s_h^j \left[\mathrm{U}b_h^{LUT}.b_h^{LUT}\right] \\ \vdots \\ s_h^{(P-1)} \left[\mathrm{U}b_h^{LUT}.b_h^{LUT}\right] \end{bmatrix} \tag{14}$$

where P is the depth of the LUT, characterized as:

$$P = \left\lceil \left\lceil \delta_h^{max} \right\rceil \cdot 2^{b_h^{LUT}} \right\rceil \tag{15}$$

and:

$$s_h^j \left[\mathrm{U}b_h^{LUT}.b_h^{LUT}\right] = e^{t_h^j} \tag{16}$$

where:

$$t_h^j = \frac{j \cdot \left\lceil \delta_h^{max} \right\rceil}{P - 1} \text{ for } j = 0, \ldots, P - 1 \tag{17}$$

The delay associated with Equation (1), D_1, can be expressed as the sum of the delay corresponding to the calculation of distance (see Equation (11)) and the delay corresponding to processing of the LUT:

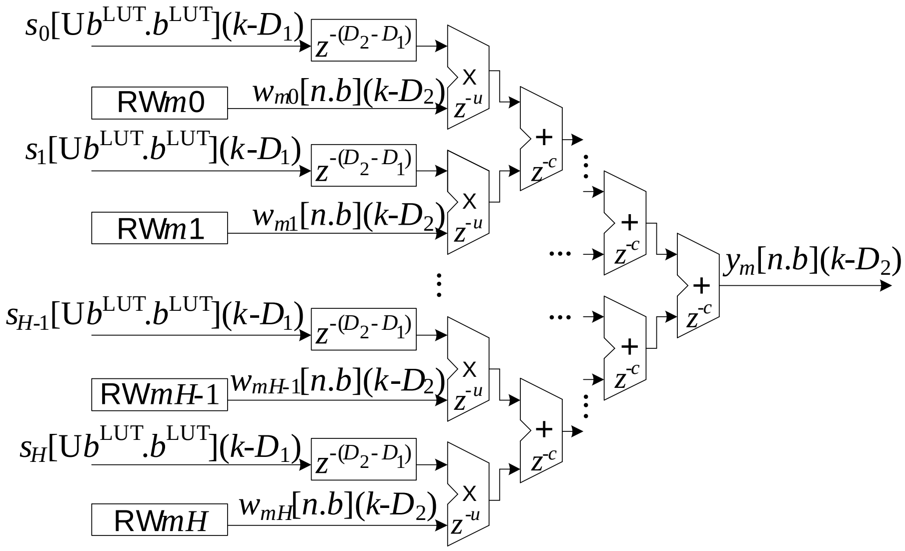
$$D_1 = r + q + D \tag{18}$$

where q is the LUT delay (z^{-q})

3.3. Output Layer Neurons

The implementation of each m-th neuron, N_m, of the output layer is illustrated in Figure 5. Similar to the processing presented in Figure 3, the sums of the products amongst the inputs and the synaptic weights were also implemented in a partially parallel manner [2,7,26].

Figure 5. The implementation of the m-th neuron, N_m, of the output layer (see Equation (6)).



In the case of the implementations associated with the output neurons, the addition operations can have delays of z^{-u} samples, and the multiplication operations can have delays of z^{-c} samples. The total accumulated delay, from the input to the output of the RBF, can be expressed by:

$$D_2 = D_1 + u + c\log_2(H) + c \tag{19}$$

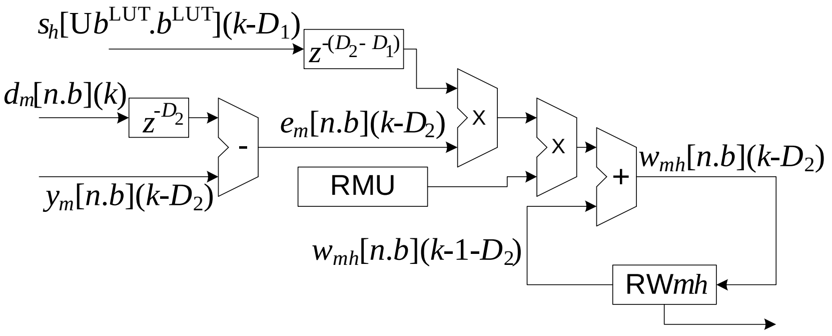
3.4. Online LMS Algorithm

The implementation of the online LMS associated with each synaptic weight, according to Equation (9), is presented in the diagram shown in Figure 6. All $M \times H$ weights are updated in parallel and stored in the registers, RWmh. The RMU register stores the value of the learning rate, μ, and to avoid problems in realigning the algorithm, the operations of addition, subtraction and multiplication are implemented with zero delay.

3.5. Delays of Operations

Due to the reduction in the size of transistors and increases in clock frequency, delays caused by interconnection paths (also known as routing delays) are one of the dominant factors affecting time restrictions. In FPGAs, routing delays are caused by programmable routing switches, which significantly increases the wire delay [27]. Techniques, such as wire pipelining (or delay padding), in which flip-flops or latches are inserted between critical stages, can reduce the paths in order to achieve the necessary time restrictions [27,28].

Figure 6. Updating of the synaptic weights, $w_{mh}\left(k\right)$, according to Equation (9).



Hence, based on the technique of wire pipelining, it can be seen from Equations (11), (18) and (19) that there are delays associated with the operations of addition, subtraction, multiplication and ROM reading in the structures presented in Figures 3–5. The delays are implemented by the addition of flip-flops or latches after the operations, represented here by the variables a, r, u, c and q. It is important to highlight that the operations (addition and multiplication) in the updating circuit of the synaptic weights, $w_{mh}\left(k\right)$ (see Figure 6), do not use delays (wire pipelining technique). This restriction ensures synchronization of updating of the weights with their corresponding inputs.

It can be seen from the case studies presented in [29–33] using Virtex-6 that the value of the delay influences the maximum clock frequency associated with the arithmetic operation in question. For example, in [29], it was observed that the operation of addition in Virtex-6 for variables of 32 bits with a signal could achieve a maximum clock frequency of 410 MHz, using a delay of z^{-3}, while a maximum clock frequency of 388 MHz was obtained in the absence of any delay.

3.6. Analysis of the Area Occupied

The occupied area (in FPGA) of the RBF network can be expressed as:

$$\begin{aligned} A^{RBF}(n,a,r,c,u,p,H,M,\alpha,\gamma,\eta,\kappa,P,D_1,D_2) &= \left\{A^{ILM}(n,a,r,p,H,P,\gamma,\alpha)\right\} \\ &\quad+\left\{A^{OLM}(n,c,u,M,H,D_1,D_2,\eta)\right\} \\ &\quad+\left\{A^{UAM}(n,M,H,\kappa,D_1,D_2)\right\} \end{aligned} \tag{20}$$

where:

$$\begin{aligned} A^{ILM}(n,a,r,p,H,\alpha,\gamma,P) &= \Big\{N^{ILM}_{FF}(n,a,r,p,H,P,\gamma,\alpha) \\ &\quad N^{ILM}_{LUT}(n,a,r,p,H,P,\gamma,\alpha) \\ &\quad N^{ILM}_{EmbMult}(p,H,\gamma), N^{ILM}_{BRAM}(n,H,\alpha,P)\Big\} \end{aligned} \tag{21}$$

$$\begin{aligned} A^{OLM}(n,c,u,M,H,\eta,D_1,D_2) &= \Big\{N^{OLM}_{FF}(n,c,u,M,H,D_1,D_2,\eta), \\ &\quad N^{OLM}_{LUT}(n,c,u,M,H,D_1,D_2,\eta) \\ &\quad N^{OLM}_{EmbMult}(M,H,\eta)\Big\} \end{aligned} \tag{22}$$

and

$$A^{UAM}(n, M, H, \kappa, D_1, D_2) = \{N_{FF}^{UAM}(n, M, H, \kappa, D_1, D_2), N_{LUT}^{UAM}(n, M, H, \kappa, D_1, D_2) \\ N_{EmbMult}^{UAM}(M, H, \kappa)\} \tag{23}$$

where A^{ILM}, A^{OLM} and A^{UAM} are collections formed by the number of flip-flops, N_{FF}, number of LUTs, N_{LUT} and number of embedded multipliers, N_{Mult}, used by modules ILM, OLM and UAM, respectively. N_{BRAM} is the number of embedded blocks RAMs, for which ILM can be used to implement the H ROMs that represent the radial function [34]. It is important to observe that the ROMs can be also implemented for logic cells.

The ILM module can be expressed by:

$$\begin{aligned} N_{FF}^{ILM}(n, a, r, p, H, P, \gamma, \alpha) \leq\ & HpN_{FF}^{Sub}(n, a) + H\left(\sum_{i=0}^{\lceil \log_2(p) \rceil - 1} 2^i\right) N_{FF}^{Add}(n, a) \\ & +\gamma(1 + H)N_{FF}^{Mult}(n, r) + HpN_{FF}^{RC}(n) + HN_{FF}^{BETA}(n) \\ & +\alpha N_{FF}^{ROM}(n, q, P) + (H - \alpha)N_{FF}^{ROM_{BRAM}}(n, q, P) \end{aligned} \tag{24}$$

where $N_{FF}^{Sub}(n, a)$ and $N_{FF}^{Add}(n, a)$ are the number of flip-flops required to implement a subtraction and addition of n bits with a delay of a, respectively. $N_{FF}^{Mult}(n, r)$ is the number of flip-flops required to implement a multiplication of n bits with a delay of r. $N_{FF}^{RC}(n)$ and $N_{FF}^{BETA}(n)$ represent the number of flip-flops required to implement each register RChp and BETAh, respectively. The variable $N_{FF}^{ROM}(n, q, P)$ is the number of flip-flops used to implement with logic cells, the h-th ROM with depth P and n bits (see Equation (15)). $N_{FF}^{ROM_{BRAM}}(n, q, P)$ represents the situation where the ROM is implemented by block RAMs. The parameter γ is the portion of the Hp multipliers (ILM) that are implemented by logic cells, and α is part of the H ROMs, which are also implemented in logic cells. The number of LUTs can be expressed by:

$$\begin{aligned} N_{LUT}^{ILM}(n, a, r, p, H, P, \gamma, \alpha) \leq\ & HpN_{LUT}^{Sub}(n, a) + H\left(\sum_{i=0}^{\lceil \log_2(p) \rceil - 1} 2^i\right) N_{LUT}^{Add}(n, a) \\ & +\gamma(1 + H)N_{LUT}^{Mult}(n, r) + HpN_{LUT}^{RC}(n) + HN_{LUT}^{BETA}(n) \\ & +\alpha N_{LUT}^{ROM}(n, q, P) + (H - \alpha)N_{LUT}^{ROM_{BRAM}}(n, q, P) \\ & +N_{LUT}^{Rout}(n) \end{aligned} \tag{25}$$

where $N_{LUT}^{Sub}(n, a)$ and $N_{LUT}^{Add}(n, a)$ are the number of LUTs required to implement a subtraction and addition of n bits with a delay of a, respectively. $N_{LUT}^{Mult}(n, r)$ is the number of LUTs required to implement a multiplication of n bits with a delay of r. $N_{LUT}^{RC}(n)$ and $N_{LUT}^{BETA}(n)$ represent the number of LUTs required to implement each register RChp and BETAh, respectively. The variable $N_{LUT}^{ROM}(n, q, P)$ is the number of LUTs used to implement, with logic cells, the h-th ROM with depth P and n bits (see Equation (15)). $N_{LUT}^{ROM_{BRAM}}(n, q, P)$ represents the situation where the ROM is implemented by block RAMs. Finally, $N_{LUT}^{Rout}(n)$ is the number of LUTs used for routing [34]. The number of embedded multipliers used can be expressed as:

$$N_{EmbMult}^{ILM}(p, H, \gamma) = (Hp - \gamma) \tag{26}$$

and the number of blocks RAMs as

$$N_{BRAM}^{ILM}(n, H, \alpha, P) = (H - \alpha) N_{BRAM}^{ROM}(n, P) \tag{27}$$

where $N_{BRAM}^{ROM}(n, P)$ is the number of blocks RAMs needed to implement each h-th ROM with depth P and n bits (see Equation (15)).

For the OLM module:

$$\begin{aligned} N_{FF}^{OLM}(n, c, u, M, H, D_1, D_2, \eta) \leq \; & M \left(\sum_{i=0}^{\lceil \log_2(H) \rceil - 1} 2^i \right) N_{FF}^{Add}(n, c) + \eta N_{FF}^{Mult}(n, u) \\ & + HMN_{FF}^{Delay}(n, D_2 - D_1) \end{aligned} \tag{28}$$

$$\begin{aligned} N_{LUT}^{OLM}(n, c, u, M, H, D_1, D_2, \eta) \leq \; & M \left(\sum_{i=0}^{\lceil \log_2(H) \rceil - 1} 2^i \right) N_{LUT}^{Add}(n, c) + \eta N_{LUT}^{Mult}(n, u) \\ & + HMN_{LUT}^{Delay}(n, D_2 - D_1) + N_{LUT}^{Rout}(n) \end{aligned} \tag{29}$$

where η represents the portion of HM multipliers (in OLM) that are implemented by logic cells. The variables $N_{FF}^{Delay}(n, D_2 - D_1)$ and $N_{LUT}^{Delay}(n, D_2 - D_1)$ represent the number of flip-flops and LUTs required to implement a delay of size $D_2 - D_1$. The number of embedded multipliers in OLM can be expressed by:

$$N_{EmbMult}^{OLM}(M, H, \eta) = (HM - \eta) \tag{30}$$

Finally, the expressions that estimate the occupied area of the UAM module are expressed as:

$$\begin{aligned} N_{FF}^{UAM}(n, M, H, D_1, D_2, \kappa) = \; & MHN_{FF}^{Sub}(n, 0) + MHN_{FF}^{Add}(n, 0) + \kappa N_{FF}^{Mult}(n, 0) \\ & + HMN_{FF}^{Delay}(n, D_2 - D_1) + HMN_{FF}^{Delay}(n, D_2) \\ & + HMN_{FF}^{RW}(n) \end{aligned} \tag{31}$$

$$\begin{aligned} N_{LUT}^{UAM}(n, M, H, D_1, D_2, \kappa) = \; & MHN_{LUT}^{Sub}(n, 0) + MHN_{LUT}^{Add}(n, 0) + \kappa N_{LUT}^{Mult}(n, 0) \\ & + HMN_{LUT}^{Delay}(n, D_2 - D_1) + HMN_{LUT}^{Delay}(n, D_2) \\ & + HMN_{LUT}^{RW}(n) + N_{LUT}^{Rout}(n) \end{aligned} \tag{32}$$

where κ represents the portion of the $2HM$ multipliers that are implemented in logic cells. $N_{FF}^{RW}(n)$ and $N_{LUT}^{RW}(n)$ represent the number of flip-flops and LUTs required to implement each register RWmh, respectively. The number of embedded multipliers in UAM can be expressed by:

$$N_{EmbMult}^{UAM}(M, H, \kappa) = (2HM - \kappa) \tag{33}$$

4. Results and Experimental Tests

Two operational scenarios were analyzed in order to validate the implementation of the RBF in an FPGA. The first scenario was a widely known problem of nonlinear classification, in which the RBF attempted to copy the functioning of the XOR gate. In the second scenario, the RBF performed interpolation of the sine function. Tables 1–3 present the parameters used in the experimental tests involving the two scenarios. The results were obtained using the System Generator development platform (Xilinx) [23] and a Virtex-6 xc6vcx240t 1ff1156 FPGA. The Virtex-6 FPGA possesses $37,680$ slices grouping $301,440$ flip-flops, $150,720$ LUTs that can be used to implement logic functions or memories and 768 DSPcells (Virtex-6 FPGA DSP48E1) with multipliers and accumulators [23,29,33]. In both scenarios, the signal sampling rate was $R_s = \frac{1}{T_s}$, where T_s is the time between the k-th samples.

Table 1. Parameters used for the XOR gate scenario.

Number of inputs (p)	1
Number of centers (H)	2
Number of neurons of the output layer (M)	1
Scaling parameter associated radial function (β)	2
Learning rate (μ)	0.3125

Table 2. Parameters used for the sine function interpolation scenario.

Number of inputs (p)	1
Number of centers (H)	4
Number of neurons of the output layer (M)	1
Scaling parameter associated radial function (β)	0.125
Learning rate (μ)	0.5

Table 3. Delays associated with the arithmetical operations used in the two scenarios.

z^{-r}	3
z^{-a}	0
z^{-q}	1
z^{-u}	0
z^{-c}	0

In the results described in this section, the operations of addition and subtraction were implemented with logic cells utilizing LogiCORE IP Adder/Subtracter v11.0 (Xilinx) [30]. All of the multiplication operations were implemented using embedded multipliers ($\gamma = \eta = \kappa = 0$) of the type Virtex-6 FPGA DSP48E1 Slice with LogiCORE IP Multiplier v11.2 [29,33], and all of the ROMs associated with the radial functions were implemented using logic cells ($\alpha = H$) with LogiCORE IP Distributed Memory Generator v6.3 [32].

In the Virtex-6 FPGA, the arithmetic operations (addition, subtraction and multiplication) do not possess restrictions related to the delays ($a \geq 0$, $r \geq 0$, $u \geq 0$ and $c \geq 0$). Meanwhile, as described in Subsection 3.5, the technique of wire pipelining can assist in the routing process, reducing the distance between the operations in order to achieve the time restrictions. In the case of the ROM, implementation in the Virtex-6 FPGA requires the use of internal memory blocks (Virtex-6 block RAMs) with $q > 0$. In the absence of this condition, $q \geq 0$ (in this case, the ROM is implemented by logic cells) [31,32]. Table 3 presents all of the delay values used in the simulations, where the values selected were based on test cases presented in [29–31], where various delay values and the maximum frequencies obtained are illustrated. For example, in [29], it was found that the multiplication operation for variables of 18 bits with a signal could achieve a maximum clock frequency of 450 MHz, using a delay of $r = 3$.

4.1. Results Obtained for Synthesis of the RBF in the FPGA

Tables 4 and 5 present the results obtained after the process of synthesis of the RBF (with the parameters presented in Tables 1–3) in the FPGA. For both scenarios, it can be clearly seen that the sampling rate and the area of occupation were highly sensitive to the number of bits. On the other hand, due to the parallelization, differences between the two scenarios were not large, using the same quantity of bits (14, 15 and 16).

Table 4. Processing speed and area occupied for different fixed point formats, in the XOR gate scenario. LUT, lookup table.

Fixed Point Format ($[n.b]$)	**Sample Rate R_s (MHz)**	**Flip-Flops and Latches**	**Logic Cells (LUTs)**	**Multipliers**
$[12.8]$	100.9285	321	$1,170$	10
$[13.9]$	84.7673	349	$2,038$	10
$[14.10]$	75.5401	376	$3,610$	10
$[15.11]$	74.9569	404	$6,284$	10
$[16.12]$	66.8717	431	$12,184$	10

Table 5. Processing speed and area occupied for different fixed point formats, in the sine function scenario.

Fixed Point Format ($[n.b]$)	**Sample Rate R_s (MHz)**	**Flip-Flops and Latches**	**Logic Cells (LUTs)**	**Multipliers**
$[14.8]$	103.4554	106	$2,370$	16
$[15.9]$	80.5477	115	$3,944$	16
$[16.10]$	75.5589	124	$6,519$	16
$[17.11]$	68.4650	133	$12,801$	16
$[18.12]$	65.4332	142	$24,176$	16

Occupation of the area used by the registers is due to the storage of fixed centers (RChi), constants (Beta$h$, RMU), synaptic weights (RWmh) and delays. This is affected to a greater extent by the architecture of the ANN (number of inputs, p, number of centers, H, and number of outputs, M) than by the number of bits. Occupation of the logic cells is related to the addition operations performed using the construction of logic functions, as well as the radial functions that were developed by means of the construction of ROM memories, as described in Section 3.2. In the latter case, the degree of precision and the architecture of the network exert a direct influence, as shown in Tables 4 and 5. The multiplication operations were synthesized in internal DSP circuits, as a result of which, the area consumed by these operations remained constant, in terms of the number of bits, and only altered in terms of structural changes in the network.

4.2. Precision of the Response

Figures 7 and 8 present the results obtained for the MSE as a function of the number of samples in the two scenarios tested. In the case of the first scenario (XOR gate) (Figure 7), the MSE was calculated using frames of 16 samples, and 40 frames were tested. For the scenario involving interpolation of the sine function (Figure 8), frames of 4096 sample were utilized to calculate the MSE, and 128 frames were tested. In both cases, the RBF implemented showed satisfactory convergence, with the best results being directly related to the number of bits, n.

Figure 7. Performance of the RBF in tests using the XOR gate scenario.

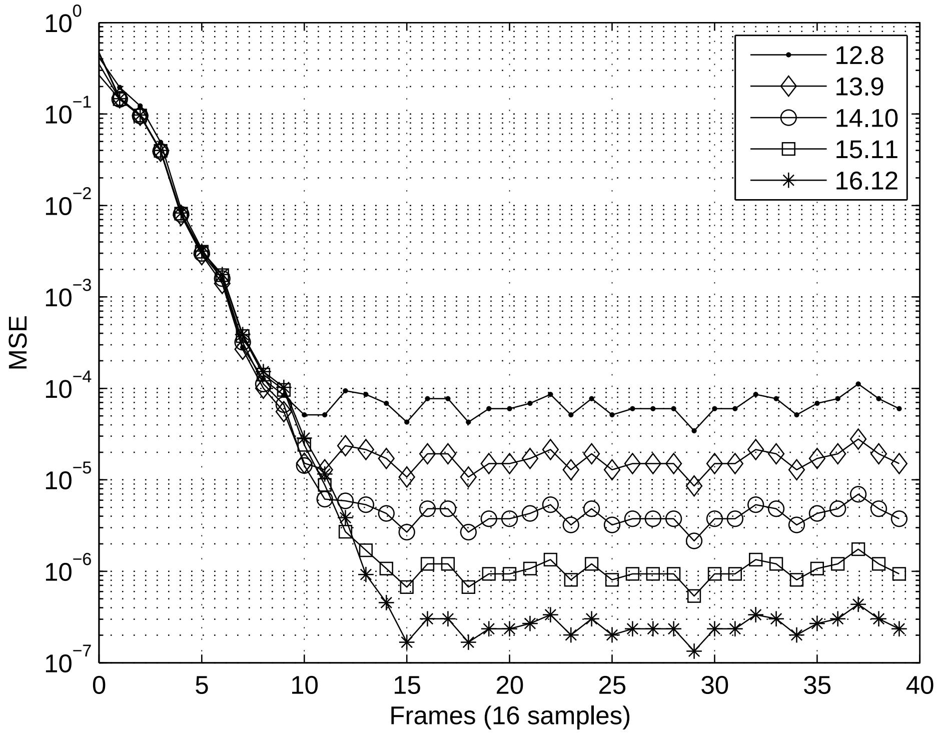
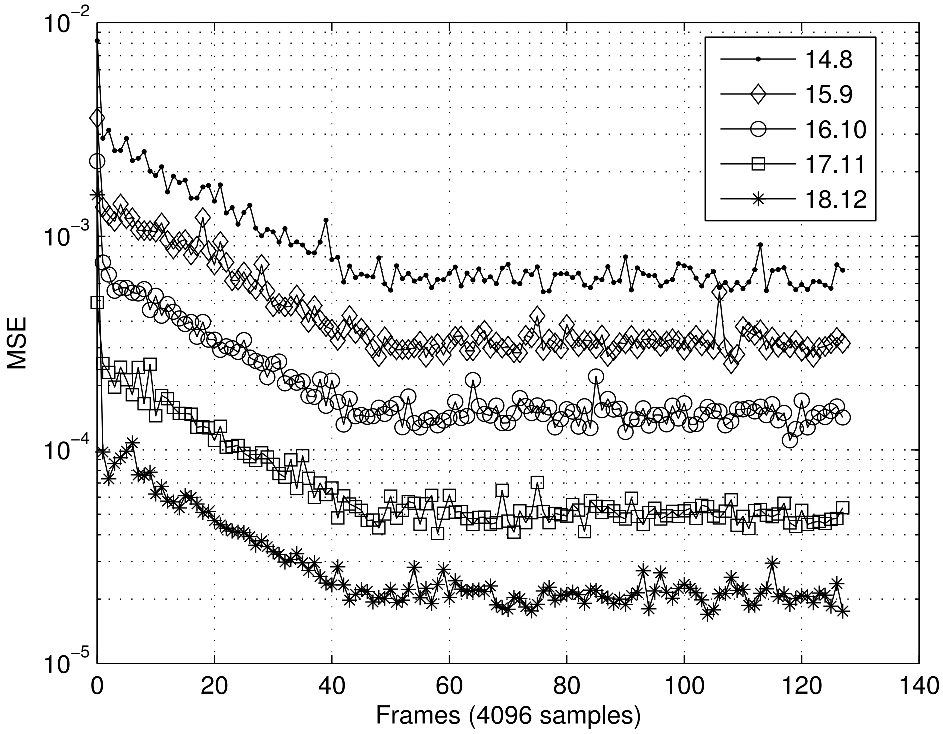


Figure 8. Performance of the RBF in tests using the sine function scenario.



4.3. Estimate of the Area Occupied

Using as a reference the parameters presented in Tables 1–3, Equations (20)–(27) and the estimates provided by the Xilinx manufacturer (see Tables 6 and 7) [29–33], it was possible to obtain estimated area occupation limits for the flip-flops and LUTs with $n = 8$ and $n = 32$ bits, as shown in Figures 9–12. These figures also show the real results obtained after the synthesis in the case of the XOR gate ($n = \{12, \ldots, 16\}$) and the sine function ($n = \{14, \ldots, 18\}$). Since embedded (fixed) multipliers were used, the multiplications did not influence the area occupied. However, it is important to point out that if the onboard multipliers are not sufficient, multipliers could be constructed with logic cells. It is important to observe that the number of bits will result in an exponential growth in the occupied area.

Table 6. Estimated number of flip-flops provided by the Xilinx manufacturer [29–33].

n	$N_{FF}^{Sub}(n,0)$	$N_{FF}^{Add}(n,0)$	$N_{FF}^{Mult}(n,3)$	$N_{FF}^{RC}(n)$	$N_{FF}^{BETA}(n)$	$N_{FF}^{RW}(n)$
8	0	0	110	8	8	8
32	91	91	–	32	32	32

n	$N_{FF}^{Delay}(n,D_2-D_1)$	$N_{FF}^{Delay}(n,D_2)$	$N_{FF}^{ROM}(n,1,P)$	$N_{FF}^{Mult}(n,0)$	$N_{LUT}^{Rout}(n)$
8	$(D_2 - D_1) \times 8$	$(D_2) \times 8$	8	110	0
32	$(D_2 - D_1) \times 32$	$(D_2) \times 32$	32	–	0

Table 7. Estimated number of LUTs provided by the Xilinx manufacturer [29–33].

n	$N_{LUT}^{Sub}(n,0)$	$N_{LUT}^{Add}(n,0)$	$N_{LUT}^{Mult}(n,3)$	$N_{LUT}^{RC}(n)$	$N_{LUT}^{BETA}(n)$	$N_{LUT}^{RW}(n)$
8	8	8	116	0	0	0
32	93	93	–	0	0	0
n	$N_{LUT}^{Delay}(n,D_1-D_2)$	$N_{LUT}^{Delay}(n,D_2)$	$N_{LUT}^{ROM}(n,1,P)$	$N_{LUT}^{Mult}(n,0)$	$N_{LUT}^{Rout}(n)$	
8	0	0	$\approx 2^{(8-4)}$	116	0	
32	0	0	$\approx 2^{(32-4)}$	–	0	

Figure 9. Area occupied (flip-flops) for different fixed point formats, in the XOR gate scenario.

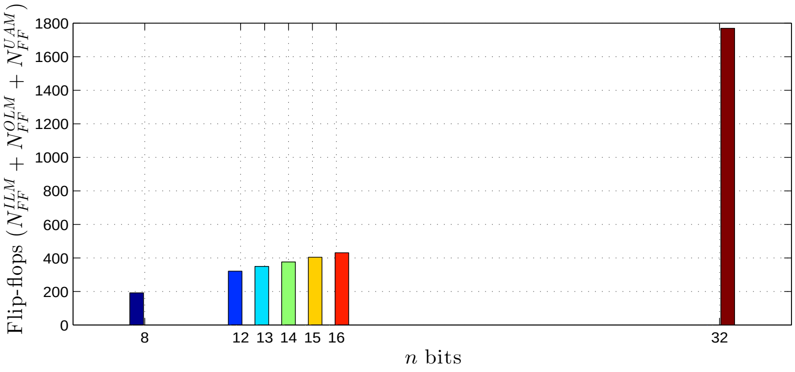


Figure 10. Area occupied (flip-flops) for different fixed point formats, in the sine function scenario.

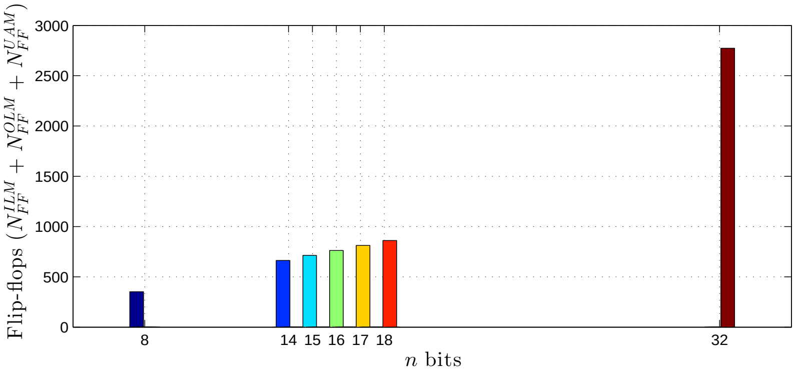


Figure 11. Area occupied (LUTs) for different fixed point formats, in the XOR gate scenario.

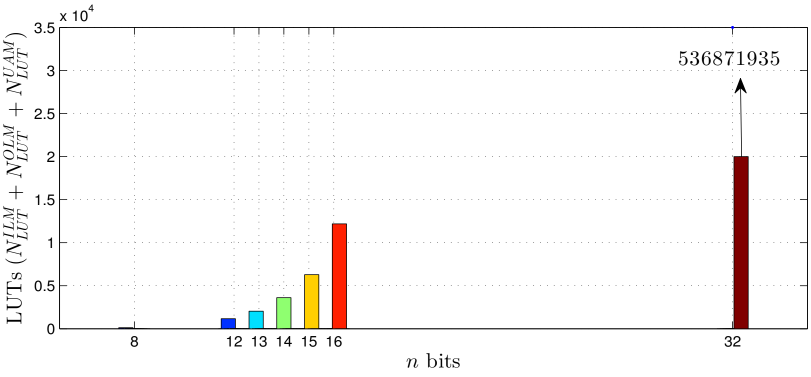


Figure 12. Area occupied (LUTs) for different fixed point formats, in the sine function scenario.

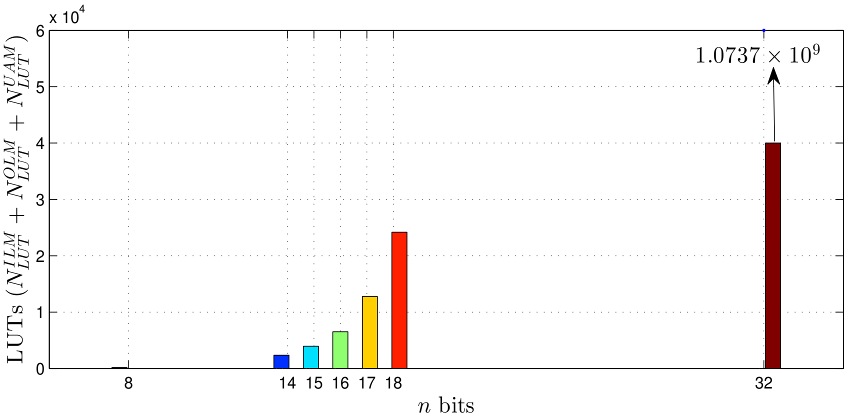


Figure 13. Area occupied (flip-flops) for different numbers of inputs, p, centers, H and $M = 1$.

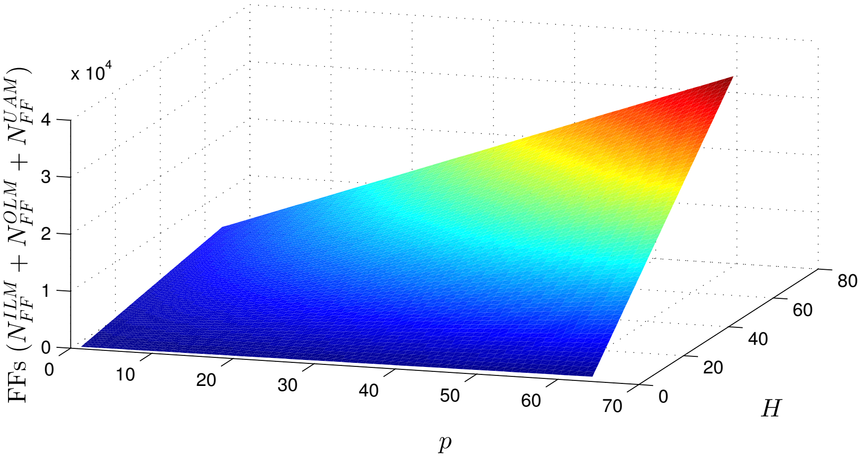
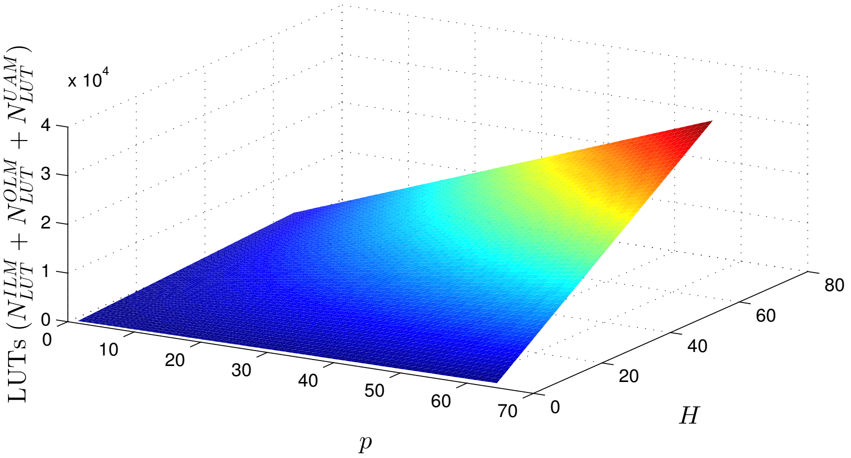


Figure 14. Area occupied (LUTs) for different numbers of inputs, p, centers, H and $M = 1$.



Figures 13 and 14 show occupation estimates for various numbers of inputs, p, and centers, H, for $M = 1$ and $n = 8$. The curves demonstrate that the occupation capacity is more influenced by the number of bits, n (exponential increase), than by the structure of the RBF that grows linearly with the parameters p, H and M. The curves illustrated in Figures 13 and 14 show that, for example, with eight bits, it is possible to work with a relatively large RBF ($p = 64$, $H = 64$ and $M = 1$) using around 50% occupancy of a Virtex 6 ($40,000$ LUTs and flip-flops more multipliers). This RBF configuration could be used in adaptive equalizers [35,36] and in some applications in mobile robotics [37].

4.4. Real-World Cases

Artificial neural networks of the type RBF can be applied to different real-world problems that may or may not require online training. Amongst these problems can be highlighted the use of RBF networks in adaptive equalizers for wireless communication systems [35,36], for obstacle avoidance in mobile robotics [37], in pattern recognition [38], as well as other applications. Nevertheless, for problems that require online training, the speed of the training in terms of the number of iterations per second is a fundamental determinant of viability in dedicated hardware. The development of the present work is based on this perspective.

Analysis of the results obtained for time (or, in other words, the sampling rate, R, of the ANN) showed that in both cases (XOR gate and sine function), the value was much more closely associated with the number of bits, n, than with the structure of the network, which was due to the parallel implementation proposed in this paper. From the results obtained, it could be seen that if space exists in the FPGA for the implementation of the ANN, the sampling rate essentially depends on the number of bits. For example, in the case of the FPGA used here, it was possible to achieve sampling rates of around 100 MHz (see Tables 4 and 5); in other words, to work at 100 mega-iterations per second.

5. Conclusions

This work presents a parallel fixed point implementation, in an FPGA, of an RBF trained with an online LMS algorithm. The implementation of the RBF was analyzed in terms of occupation area, bit resolution and processing delay. The proposed structure was tested at different resolutions, using two widely known scenarios, namely the problem of nonlinear classification with the XOR gate and the problem of interpolation utilizing the sine function. The results obtained were highly satisfactory, indicating the potential feasibility of the technique for use in practical situations of greater complexity.

Author Contributions

Marcelo A. C. Fernandes conceived and designed the experiments; Alisson C. D. de Souza performed the experiments; Marcelo A. C. Fernandes and Alisson C. D. de Souza analyzed the data; Marcelo A. C. Fernandes wrote the paper.

Conflicts of Interest

The authors declare no conflicts of interests.

References

1. Haykin, S.S. *Neural Networks: A Comprehensive Foundation*, 2nd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 1999.
2. Savich, A.; Moussa, M.; Areibi, S. The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study. *IEEE Trans. Neural Netw.* **2007**, *18*, 240–252.
3. Misra, J.; Saha, I. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing* **2010**, *74*, 239–255.
4. Dias, F.M.; Antunes, A.; Mota, A.M. Artificial neural networks: A review of commercial hardware. *Eng. Appl. Artif. Intell.* **2004**, *17*, 945–952.
5. Fan, Z.C.; Hwang, W.J. Efficient VLSI Architecture for Training Radial Basis Function Networks. *Sensors* **2013**, *13*, 3848–3877.
6. Latino, C.; Moreno-Armendariz, M.; Hagan, M. Realizing general MLP networks with minimal FPGA resources. In Proceedings of the International Joint Conference on Neural Networks, 2009 (IJCNN 2009), Atlanta, GA, USA, 14–19 June 2009; pp. 1722–1729.
7. Hariprasath, S.; Prabakar, T.N. FPGA implementation of multilayer feed forward neural network architecture using VHDL. In Proceedings of the 2012 International Conference on Computing, Communication and Applications (ICCCA), Dindigul, Tamilnadu, India, 22–24 February 2012; pp. 1–6.
8. Hassan, A.A.; Elnakib, A.; Abo-Elsoud, M. FPGA-based neuro-architecture intrusion detection system. In Proceedings of the International Conference on Computer Engineering Systems, 2008 (ICCES 2008), Cairo, Egypt, 25–27 November 2008; pp. 268–273.
9. Liddicoat, A.A.; Slivovsky, L.A.; McLenegan, T.; Heyer, D. FPGA-based artificial neural network using CORDIC modules. *Proc. SPIE* **2006**, *6313*, doi:10.1117/12.682529.
10. Orlowska-Kowalska, T.; Kaminski, M. FPGA Implementation of the Multilayer Neural Network for the Speed Estimation of the Two-Mass Drive System. *IEEE Trans. Ind. Inform.* **2011**, *7*, 436–445.
11. Mahmoodi, D.; Soleimani, A.; Khosravi, H.; Taghizadeh, M. FPGA Simulation of Linear and Nonlinear Support Vector Machine. *J. Softw. Eng. Appl.* **2011**, *4*, 320–328.
12. Patra, J.; Devi, T.; Meher, P. Radial basis function implementation of intelligent pressure sensor on field programmable gate array. In Proceedings of the 2007 6th International Conference on Information, Communications Signal Processing, Singapore, 10–13 December 2007; pp. 1–5.
13. Brassai, S.; Bako, L.; Pana, G.; Dan, S. Neural control based on RBF network implemented on FPGA. In Proceedings of the 11th International Conference on Optimization of Electrical and Electronic Equipment, 2008 (OPTIM 2008), Brasov, Romania, 22–24 May 2008; pp. 41–46.
14. Kim, J.S.; Jung, S. Evaluation of embedded RBF neural chip with back-propagation algorithm for pattern recognition tasks. In Proceedings of the 6th IEEE International Conference on Industrial Informatics, 2008 (INDIN 2008), Daejeon, Korea, 13–16 July 2008; pp. 1110–1115.
15. Kung, Y.S.; Wang, M.S.; Chuang, T.Y. FPGA-based self-tuning PID controller using RBF neural network and its application in X-Y table. In Proceedings of the IEEE International Symposium on Industrial Electronics, 2009 (ISIE 2009), Seoul, Korea, 5–8 July 2009; pp. 694–699.

16. Seob Kim, J.; Jung, S. Implementation of the RBF neural chip with the on-line learning back-propagation algorithm. In Proceedings of the IEEE International Joint Conference on Neural Networks, 2008 (IJCNN 2008), (IEEE World Congress on Computational Intelligence), Hong Kong, China, 1–8 June 2008; pp. 377–383.
17. Evert, P.; Amudhan, R.; Paul, P. Implementation of neural network based controller using Verilog. In Proceedings of the 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), Thuckafay, India, 21–22 July 2011; pp. 353–357.
18. Gargouri, A.; Krid, M.; Masmoudi, D. Hardware implementation of pulse mode RBFNN based edge detection system on virtex V platform. In Proceedings of the 2010 7th International Multi-Conference on Systems Signals and Devices (SSD), Amman, Jordan, 27–30 June 2010; pp. 1–5.
19. Vizitiu, I.C.; Rincu, I.; Radu, A.; Nicolaescu, I.; Popescu, F. Optimal FPGA implementation of GARBF systems. In Proceedings of the 2010 12th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM), Basov, Ukraine, 20–22 May 2010; pp. 774–779.
20. Kim, J.S.; Jung, S. Joint control of ROBOKER arm using a neural chip embedded on FPGA. In Proceedings of the IEEE International Symposium on Industrial Electronics, 2009 (ISIE 2009), Seoul, Korea, 5–8 July 2009; pp. 1007–1012.
21. Yang, F.; Paindavoine, M. Implementation of an RBF neural network on embedded systems: Real-time face tracking and identity verification. *IEEE Trans. Neural Netw.* **2003**, *14*, 1162–1175.
22. Tisan, A.; Cirstea, M. SOM neural network design—A new Simulink library based approach targeting FPGA implementation. *Math. Comput. Simul.* **2013**, *91*, 134–149.
23. Xilinx System Generator User's Guide. Available online: http://www.xilinx.com (accessed on 26 September 2014).
24. Matlab/Simulink. Available online: http://www.mathworks.com (accessed on 26 September 2014).
25. Haykin, S.S. *Adaptive Filter Theory*, 3rd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 1996.
26. Al-Kazzaz, S.; Khalil, R. FPGA Implementation of Artificial Neurons: Comparison study. In Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications, 2008 (ICTTA 2008), Damascus, Syria, 7–11 April 2008; pp. 1–6.
27. Singhal, L.; Bozorgzadeh, E.; Eppstein, D. Interconnect Criticality-Driven Delay Relaxation. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2007**, *26*, 1803–1817.
28. Dong, X.; Lemieux, G. PGR: Period and glitch reduction via clock skew scheduling, delay padding and GlitchLess. In Proceedings of the International Conference on Field-Programmable Technology, 2009 (FPT 2009), Sydney, Australia, 9–11 December 2009; pp. 88–95.
29. Xilinx—LogiCORE IP Multiplier V11.2—Datasheet. Available online: http://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf (accessed on 26 September 2014).
30. Xilinx—LogiCORE IP Adder/Subtracter V11.0—Datasheet. Available online: http://www.xilinx.com/support/documentation/ip_documentation/addsub_ds214.pdf (accessed on 26 September 2014).

31. Xilinx—LogiCORE IP Block Memory Generator v6.3—Datasheet. Available online: http://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v6_3/blk_mem_gen_ds512.pdf (accessed on 26 September 2014).
32. Xilinx—LogiCORE IP Distributed Memory Generator v6.3—Datasheet. Available online: http://www.xilinx.com/support/documentation/ip_documentation/dist_mem_gen/v6_3/dist_mem_gen_ds322.pdf (accessed on 26 September 2014).
33. Xilinx—Virtex 6 FPGA DSP48E1 Slice—User Guide. Available online: http://www.xilinx.com/support/documentation/user_guides/ug369.pdf (accessed on 26 September 2014).
34. Kilts, S. *Advanced FPGA Design: Architecture, Implementation, and Optimization*; Wiley: Hoboken, NJ, USA, 2007.
35. Burse, K.; Yadav, R.; Shrivastava, S. Channel Equalization Using Neural Networks: A Review. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2010**, *40*, 352–357.
36. Katz, G.; Sadot, D. A nonlinear electrical equalizer with decision feedback for OOK optical communication systems. *IEEE Trans. Commun.* **2008**, *56*, 2002–2006.
37. Cinar, E.; Sahin, F. EOG controlled mobile robot using Radial Basis Function Networks. In Proceedings of the Fifth International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control, 2009 (ICSCCW 2009), Famagusta, Cyprus, 2–4 September 2009; pp. 1–4.
38. Lampariello, F.; Sciandrone, M. Efficient training of RBF neural networks for pattern recognition. *IEEE Trans. Neural Netw.* **2001**, *12*, 1235–1242.