Proposta de Implementação Paralela em Ponto Fixo de uma Rede de Funções Radias de Base para FPGA

Alisson C. D. de Souza* e Marcelo A. C. Fernandes†
Departamento de Engenharia da Computação e Automação – DCA
Centro de Tecnologia – CT
Universidade Federal do Rio Grande do Norte – UFRN
Natal, Brasil
*alisson.camara@dca.ufrn.br, †mfernandes@dca.ufrn.br

***Resumo*—Este artigo propõe uma implementação paralela em ponto fixo para FPGA (*Field-programmable gate array*) de uma Rede Neural Artificial (RNA) do tipo Funções Radiais de Base (RBF - *Radial Basis Function*) treinada com o algoritmo do LMS (*Least Mean Square*). Resultados associados com o tempo de processamento e a área de ocupação para vários formatos em ponto-fixo são analisados. Estudos relativos à precisão da resposta da RNA para o problema de classificação não linear com a porta XOR e o problema de interpolação utilizando a função Seno, também foram analisados para implementação em hardware. Todo projeto foi desenvolvido utilizando a plataforma de desenvolvimento *System Generator* da Xilinx tendo como FPGA alvo uma Virtex 6 xc6vcx240t-1ff1156.**

***Palavras-chave*—Redes Neurais, RBF, FPGA, Ponto Fixo.**

***Abstract*—This paper proposes a parallel implementation of the radial basis function (RBF) (trained with the LMS algorithm) in fixed point for field-programmable gate array (FPGA). Results associated with the processing time and area occupancy (in FPGA) for various fixed-point formats are analyzed. Studies concerning the accuracy of the RNA response for the nonlinear classification problem with XOR gate and the interpolation problem using the Sine function, were also analyzed for the hardware implementation. The project was developed using the System Generator software (Xilinx development platform) and the Virtex-6 xc6vcx240t 1ff1156 FPGA.**

***Keywords*—Neural Networks, RBF, FPGA, Fixed Point.**

I. Introdução

As Redes Neurais Artificiais (RNA) são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neural dos organismos inteligentes e que adquirem conhecimento através de experiência passada e presente. Esses organismos inteligentes são formados por um conjunto extremamente complexo de células chamadas de neurônios biológicos; já a estrutura de uma RNA é composta por várias unidades de processamento, chamadas de neurônios artificiais, que funcionam interconectados de forma paralela e distribuída [1]. Uma das arquiteturas mais utilizadas de RNA são as redes de Funções Radiais de Base (RBF - *Radial Basis Function*) associadas ao algoritmo do LMS (*Least Mean Square*) em seu treinamento.

Pode-se observar que do ponto de vista de implementação, um dos grandes problemas da RBF é a falta de metodologia para definição de sua topologia em termos de número de centros, que por outro lado impacta positivamente em seu custo computacional. Existem algumas heurísticas como apresentadas em [1] que direcionam o projetista na topologia da rede. No entanto, há um consenso é que esta topologia é dependente do problema em questão [2]. Uma prática bastante comum é testar várias topologias para vários conjuntos de informação, porém esta prática requer bastante tempo e, por outro lado, pode ser aplicada apenas em casos de treinamento off-line, ou quando a informação associada à estatística sobre o espaço de dados é conhecida.

Objetivando acelerar o funcionamento e o treinamento das RNA, várias soluções para implementação de circuitos integrados dedicados (ASIC - *Application Specific Integrated Circuit*) são propostos em [3]. Porém, as implementações em ASIC fixam a arquitetura e o algoritmo implementado dando pouca flexibilidade ou uma flexibilidade associada a um custo elevado. Todavia, com o avanço das estruturas de hardware reconfigurável as implementações das RNA's voltam-se ao foco das pesquisas onde agora, pode-se ter estruturas em hardware dedicadas e flexíveis quanto a sua topologia e algoritmo de treinamento. Atualmente, arquiteturas mais utilizadas para hardware reconfigurável são as FPGA's que podem proporcionar uma performance e densidade semelhante ao um ASIC com a vantagem de utilização de prototipagem rápida. Em [2] – [6] são apresentados vários estudos e propostas de implementação do Perceptron de Múltiplas Camadas (MLP - *Multilayer Perceptron*) em FPGA. Já estudos associados à implementação da RBF em FPGA podem ser encontrados em [7] – [15].

Assim, este trabalho apresenta uma proposta de implementação paralela em hardware de uma RNA do tipo RBF. A implementação é feita em ponto-fixo e têm como alvo arquiteturas de hardware reconfigurável do tipo FPGA's. Diferentemente das propostas apresentadas na literatura [2] – [15], este artigo faz uma análise detalhada da implementação da RBF apresentando aspectos relacionados ao tempo de processamento, ao atraso, a precisão da resposta e a implementação da função radial de base. As análises são feitas para dois cenários bastante conhecidos na literatura. O primeiro é caracterizado pelo problema da classificação não-linear associado à porta XOR e o segundo representa um problema de interpolação com a função Seno. Todos os resultados foram obtidos utilizando a plataforma de desenvolvimento *Xlinx - System Generator* [16] e uma FPGA Virtex 6 xc6vcx240t 1ff1156.

II. Redes de Funções Radiais de Base

As redes RBF têm tido recentemente uma significativa posição dentro do domínio das redes neurais artificiais [1]. A principal razão para esse resultado é a simplicidade do processo de treinamento e a eficiência computacional quando comparada a MLP [1]. A estrutura da rede RBF é do tipo múltiplas camadas, o processamento é do tipo *feedforward* e o treinamento pode ser supervisionado, método que foi aplicado neste trabalho, ou híbrido, no qual combina-se um método não supervisionado com um supervisionado.

A. Arquitetura

A estrutura básica de uma rede RBF, apresentada na Figura 1, consiste de apenas três camadas. A primeira camada é a conexão do modelo como o meio, formada por p entradas. A segunda camada ou camada escondida é formada por H funções radiais de base (também chamadas de neurônios) e realiza uma transformação não-linear do espaço vetorial de entrada para um espaço vetorial interno que geralmente tem uma dimensão maior ($H > p$). A última camada, a camada de saída, transforma o espaço vetorial interno em uma saída, através de um conjunto de M neurônios lineares.

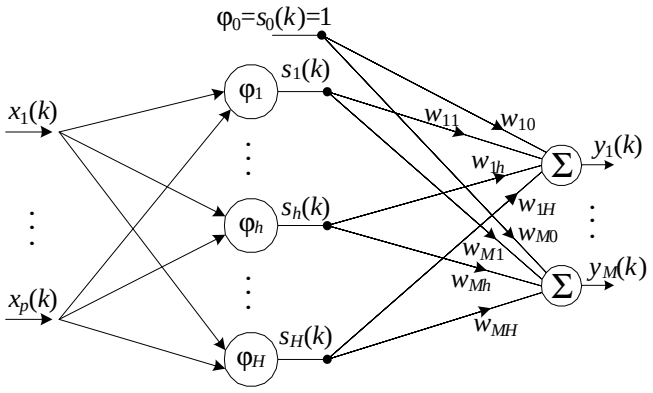

Figura 1. Arquitetura da RBF.

As funções radiais de base produzem um resposta significativa, diferente de zero, somente quando o padrão de entrada está dentro de uma região pequena localizada no domínio da função. Cada função requer um centro e um parâmetro escalar, β, e a função que é mais utilizada como função radial de base é a Gaussiana [1]. Assim, a h-ésima função, $\varphi_h(\cdot)$, pode ser expressa por

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

onde $\beta = \frac{1}{2\sigma^2}$ e $v_h(k)$, que representa a distância da entrada, $\mathbf{x}(k)$, em relação ao h-ésimo centro, $\mathbf{c}_h$, é expressa como

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

O vetor de centros associada a h-ésima função radial é caracterizado como

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

é o vetor de entradas como

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

Assim, a Equação 2 pode ser rescrita e expressa por

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

A saída do m-ésimo neurônio, N_m, da camada de saída pode ser caracterizada como

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

Substituindo a Eq. (1) na Eq. (6), tem-se

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

onde w_{mh} é o peso sináptico entre o neurônio h da camada escondida com o neurônio m da camada de saída. Em cada instante k a rede RBF recebe um vetor de entradas $\mathbf{x}(k)$ e gera como saída um saída um vetor $\mathbf{y}(k)$ expresso por

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

B. Algoritmo de Treinamento

Devido à diferença entre a camada escondida e a camada de saída, o treinamento das redes RBF é dividido geralmente em duas partes. A primeira parte é relativa à otimização não-linear feita pela camada escondida. Nessa o vetor de entrada, $\mathbf{x}(k)$, é processado através das funções presentes na camada escondida, caracterizadas pela Equação 1. Existem várias estratégias para se determinar os centros, $\mathbf{c}_h$, [1]. A estratégia utilizada nesse trabalho, seleciona centros fixos de forma determinística, diferente da forma convencional, onde são selecionados centros fixos de forma aleatória [1]. Na segunda parte do treinamento calculam-se os pesos, w_{mh} entre a camada escondida e a camada de saída. A obtenção dos pesos pode ser feita através do método da pseudo inversa [1] ou do algoritmo do LMS [17] que foi o método utilizado neste trabalho. O LMS é uma técnica iterativa que otimiza a função de erro quadrático médio (MSE - *Mean Squared Error*) pelo método do gradiente descendente com a clássica aproximação estocástica (substitui a esperança matemática por uma estimativa instantânea) [17]. Os parâmetros são ajustados a cada instante k de acordo com a seguinte expressão

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

onde μ é o passo de adaptação e $e_m\,(k)$ é o erro do treinamento associado ao m–ésimo neurônio de saída no k-ésimo instante. O erro pode ser expresso por

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

onde $d_m(k)$ é valor desejado para o m–ésimo neurônio de saída. O LMS é menos complexo computacionalmente quando comparado ao método da pseudo inversa que necessita calcular a inversão de uma matriz não quadrada para obtenção dos pesos, $w_{mh}\,(k)$ [17].

III. Implementação e Projeto

A. Estrutura Geral

A Figura 2 apresenta a arquitetura geral da implementação proposta. Todas as variáveis e constantes estão em ponto fixo utilizando uma resolução de *n* bits dos quais *b* bits representam a parte fracionária e $(n-b)$ bits representam a parte inteira. Para variáveis em ponto fixo com sinal é utilizada a representação $[n.b]$ e para as variáveis sem sinal a reapresentação é caracterizada como $[\mathrm{U}n.b]$. A arquitetura é formada por três grandes módulos que são caracterizados como camada intermediária, camada de saída e algoritmo de atualização, como ilustrado na Figura 3.

1) *Módulo Camada Intermediária* (MCI): Consiste do cálculo das funções radiais de base de acordo com a equação 1 para gerar o sinal $s_h(k)$ a partir da entrada, $\mathbf{x}(k)$. O processamento das H funções radiais, $\varphi_h\,(\cdot)$, é feito de forma paralela a partir dos centros armazenados nos registradores locais de n bits. Cada centro, $c_{h,i}$ (ver Equação 3) é armazenado no registrador RChi, como ilustrado na Figura 2.
2) *Módulo Camada Saída* (MCS): Representa o processamento feito pelos M neurônios de saída como apresentado na Equação 6. Neste módulo, os M nerônios também funcionam em paralelo, gerando o sinal $\mathbf{y}(k)$ (ver Equação 8) a partir da entrada, $\mathbf{x}(k)$.
3) *Módulo Algoritmo de Atualização* (MAA): Este módulo é responsável por implementar o algoritmo de atualização, que no caso deste trabalho é o LMS. A atualização de cada peso sináptico, $w_{mh}(k)$, no k-ésimo instante é feito em paralelo de acordo com a Equação 9. Como apresentado na Figura 2, os pesos w_{mh} são armazenados em registradores locais de n bits, chamados aqui de RWmh.

Os módulos MCI, MCS e MAA funcionam de forma sequencial e, para cada instante k, o vetor de entradas, $\mathbf{x}(k)$, é processado gerando com saída o vetor $\mathbf{y}(k)$ e todos os pesos sinápticos são calculados atualizando os registradores RWmh. É importante observar que ainda existe um espaço para melhorar o desempenho da RBF, caso os módulos funcionem em *pipeline*, situação que não foi implementada neste trabalho.

B. Funções Radiais de Base

As Figuras 3 e 4 apresentam em detalhes o processamento associado a h-ésima função radial de base do MCI.

A Figura 3 detalha a implementação referente a Equação 5. Neste caso a implementação do somatório é realizada de forma parcialmente paralela como apresentado em [2], [5] e

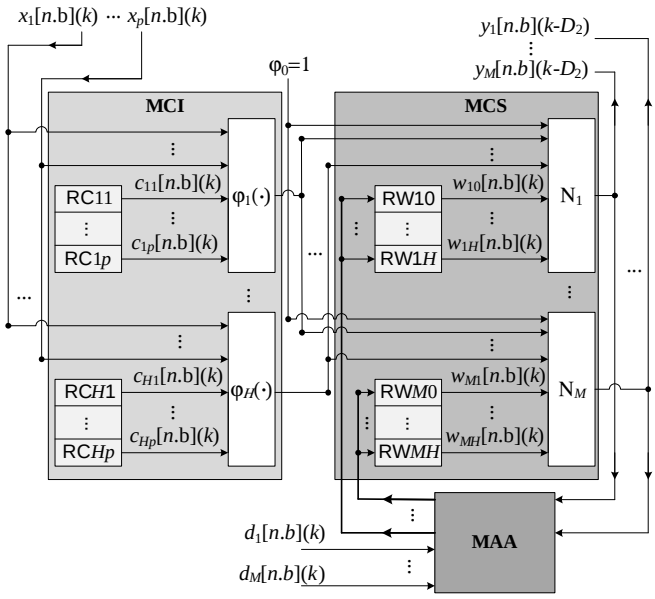

Figura 2. Estrutura geral da rede RBF implementada em FPGA.

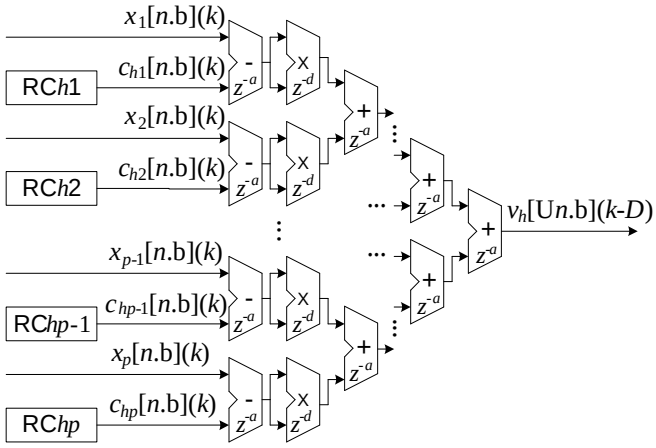

Figura 3. Implementação referente as Equações 2 e 5.

[18]. Os atrasos associados aos somadores e multiplicadores são também apresentados na Figura 3. Cada somador pode ser implementado com um atraso z^{-a} e cada multiplicado com um atraso z^{-d}. A inserção de atrasos as operações relaxam as condições de roteamento entre as células na FPGA, principalmente em operações complexas como o caso dos multiplicadores. Por outro lado, a introdução dos atrasos, retardam a resposta do sistema que em muitos casos não interessante [16]. O cálculo do atraso total, D, relativo a Equação 5 pode ser expresso como

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

Existem várias técnicas para o cálculo de funções não lineares em hardware e, entre elas, uma das mais utilizadas para FPGA's é a aproximação da função por uma tabela de valores chamada de *Lookup Table* (LUT) [2], [5], [18]. A Figura 4 detalha a implementação proposta para h-ésima função radial (ver Equação 1), no qual foi utilizada uma

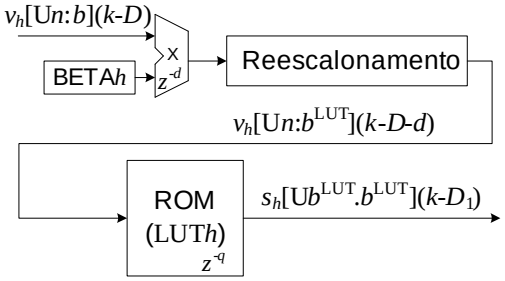

Figura 4. Implementação referente a Equação 1.

ROM (*Read-only Memory*) para o desenvolvimento da LUT. Para cada h-ésima função radial existe uma LUTh e uma operação de produto para o ajuste do parâmetro escalar β (armazenado no registrador BETAh). Para obter uma maior resolução associada a cada h-ésima LUTh a variável $v_h(k)$ pode ser reescalonada para um novo formato expresso por $\left[\mathrm{U}n.b_h^{LUT}\right]$, no qual, b_h^{LUT} é um novo valor para o número bits da parte fracionária que pode ser calculado da seguinte forma

$$b_h^{LUT} = n - \lceil \log_2 (\lceil d_h^{max} \rceil) \rceil \,, \tag{12}$$

onde

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

A Equação 13 determina a maior distância de todos os conjuntos de entrada ao h-ésimo centro e, com esta informação, pode-se reduzir o número de bits da parte inteira em $\left(n - b_h^{LUT}\right)$ e aumentar o número de bits da parte fracionária em b_h^{LUT}. A Figura 4 ilustra a etapa de reescalonamento após a operação de produto e antes da LUT.

Como $v_h(k) \cdot \beta > 0$, para qualquer k o valor da resposta da função radial é limitada em $0 \leq s_h(k) \leq 1$ podendo, então, ser representada em ponto fixo como $\left[\mathrm{U}b_h^{LUT}.b_h^{LUT}\right]$, no qual o número de bits da parte inteira é zero. Os valores armazenados na h-ésima LUT podem ser caracterizados pelo vetor LUT_h que é expresso como

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

onde P é a profundidade da LUT que pode ser caracterizada como

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

e

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

onde

$$t_h^j = \frac{j \cdot \lceil d_h^{max} \rceil}{P-1} \text{ para } j = 0, \ldots, P-1. \tag{17}$$

O atraso associado a equação 1, D_1, pode ser expresso como a soma do atraso referente ao cálculo da distância (ver Equação 11) e o atraso referente ao processamento da LUT, ou seja,

$$D_1 = d + q + D. \tag{18}$$

C. Neurônios da Camada de Saída

A implementação de cada m-ésimo neurônio, N_m, da camada de saída é apresentado no esquema da Figura 5. Semelhante ao processamento apresentado na Figura 3, a soma dos produtos entre entradas e os pesos sinápticos também foram implementados de maneira parcialmente paralela [2], [5], [18].

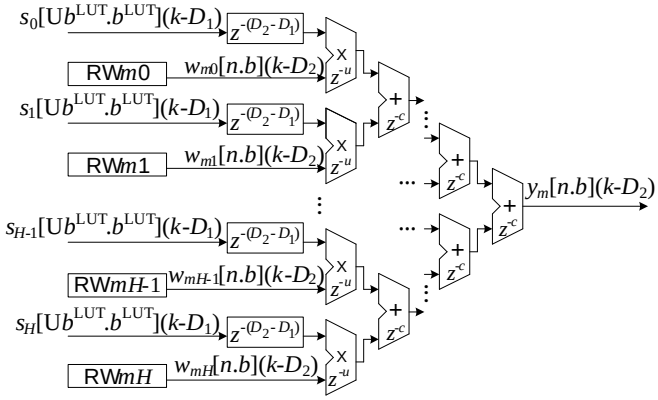

Figura 5. Implementação do m-ésimo neurônio, N_m, da camada de saída (ver Equação 6).

Para o caso das implementações associadas aos neurônios de saída, as operações de soma poderão ter atrasos de z^{-u} amostras e as operações de multiplicação poderão ter atrasos de z^{-c} amostras. O atraso total acumulado da entrada até a saída da RBF pode ser expresso por

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

D. LMS

A implementação do LMS associada a cada peso sináptico de acordo com a equação 9 é apresentada no diagrama da Figura 6. Todos $M \times H$ pesos são atualizados paralelamente e armazenados nos registradores RWmh. O registrador RMU armazena o valor do passo de adaptação, μ, e para evitar um problema na realimentação do algoritmo, as operações de adição, subtração e multiplicação são implementadas com atraso zero.

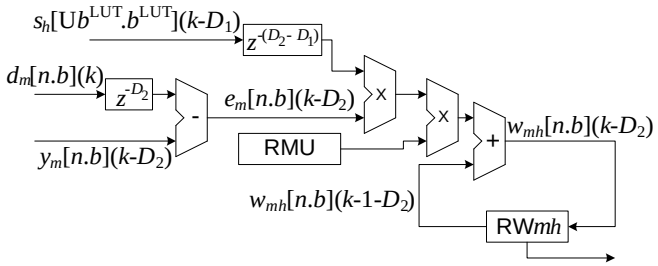

Figura 6. Atualização do peso sináptico, $w_{mh}(k)$, de acordo com a equação 9.

IV. Resultados e Testes Experimentais

Objetivando validar a proposta de implementação da RBF em FPGA foram analisados dois cenários de funcionamento.

O primeiro cenário é caracterizado como um problema de classificação não-linear amplamente conhecido no qual, a RBF tenta copiar o funcionamento da porta XOR. Já no segundo cenário a RBF irá realizar a interpolação da função Seno. As Tabelas I, II e III apresentam os parâmetros utilizados nos testes experimentais para os dois cenários. Os resultados foram obtidos utilizando a plataforma de desenvolvimento *Xlinx - System Generator* [16] e uma FPGA Virtex 6 xc6vcx240t 1ff1156. A FPGA Virtex 6 utilizada possui 37.680 *slices* que agrupam 301.440 registradores (*flip-flops*), 150.720 células lógicas que podem ser utilizados para implementar funções lógicas ou memórias e 768 células de DSP com multiplicadores e acumuladores [16]. Nos dois cenários implementados os sinais trabalham em uma taxa de amostragem, $R_s = \frac{1}{T_s}$ onde T_s é o tempo entre as k-ésimas amostras.

Tabela I. PARÂMETROS UTILIZADOS PARA O CENÁRIO ASSOCIADO A PORTA XOR

Número de entradas (p)	1
Número de centros (H)	2
Número de neurônios da camada de saída (M)	1
Parâmetro escalar associado a função radial (β)	2
Passo de adaptação (μ)	$0,3125$

Tabela II. PARÂMETROS UTILIZADOS PARA O CENÁRIO DE INTERPOLAÇÃO DA FUNÇÃO SENO

Número de entradas (p)	1
Número de centros (H)	4
Número de neurônios da camada de saída (M)	1
Parâmetro escalar associado a função radial (β)	$0,125$
Passo de adaptação (μ)	$0,5$

Tabela III. ATRASOS ASSOCIADOS AS OPERAÇÕES ARITMÉTICAS UTILIZADAS NOS DOIS CENÁRIOS.

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

A. Resultados Relativos a Síntese

As Tabelas IV e V apresentam os resultados obtidos após o processo de síntese da RBF (parâmetros apresentados nas Tabelas I, II e III) na FPGA. Observa-se claramente, para ambos os cenários, que a taxa de amostragem e área de ocupação são parâmetros muito sensíveis ao número de bits. Por outro lado, devido a paralelização a variação entre os dois cenários para a mesma quantidade de bits (14, 15 e 16) não é tão grande.

Tabela IV. VELOCIDADE DE PROCESSAMENTO E ÁREA OCUPAÇÃO PARA VÁRIOS FORMATOS DE PONTO FIXO EM RELAÇÃO AO CENÁRIO DA PORTA XOR

Formato ($[n.b]$)	Taxa Amostragem R_s (MHz)	Registradores (*Flip-Flops*)	Células Lógicas	Multiplicadores
$[12.8]$	$100,9285$	54	620	6
$[13.9]$	$84,7673$	59	1185	6
$[14.10]$	$75,5401$	64	2000	6
$[15.11]$	$74,9569$	69	3989	6
$[16.12]$	$66,8717$	74	7909	6

Tabela V. VELOCIDADE DE PROCESSAMENTO E ÁREA OCUPAÇÃO PARA VÁRIOS FORMATOS DE PONTO FIXO EM RELAÇÃO AO CENÁRIO DA FUNÇÃO SENO

Formato ($[n.b]$)	Taxa de Amostragem R_s (MHz)	Registradores (*Flip-Flops*)	Células Lógicas	Multiplicadores
$[14.8]$	$103,4554$	106	1244	8
$[15.9]$	$80,5477$	115	2269	8
$[16.10]$	$75,5589$	124	3962	8
$[17.11]$	$68,4650$	133	8126	8
$[18.12]$	$65,4332$	142	15686	8

A ocupação de área gasta em registradores é devido ao armazenamento dos centros fixos (RChi), das constantes (Beta$h$, RMU), dos pesos sinápticos (RWmh) e dos atrasos, que é mais relacionada a arquitetura (número de entradas, p, número de centros, H, e número de saídas ,M) da RNA do que o número de bits. Já a ocupação das células lógicas é relacionada as operações de soma que foram implementadas através da construção de funções lógicas e também a implementação das funções radiais que foram desenvolvidas através da construção memórias do tipo ROM como apresentado na Seção III-B. Para este último caso o tamanho da precisão e a arquitetura da rede influência de forma direta como apresentado nas Tabelas IV e V. As operações de multiplicação foram sintetizadas nos circuitos de DSP's internos e, por este motivo, o consumo de área destas operações se mantiveram constantes em relação ao número de bits, sofrendo alterações apenas em relação as mudanças estruturais da rede.

B. Resultados Associados à Precisão da Resposta

As Figuras 7 e 8 apresentam os resultados obtidos para o MSE em função do número de amostras nos dois cenários testados. Para o primeiro cenário (caso da porta XOR), ilustrado na Figura 7, o MSE foi calculado em quadros de 16 amostras e foram testados 40 quadros. Para o cenário com interpolação da função Seno, ilustrado na Figura 8, foram utilizados quadros de 2048 amostras para o cálculo do MSE e foram testados 64 quadros. Em ambos os testes, a RBF implementada apresentou resultados satisfatórios de convergência, no qual os melhores resultados estão diretamente relacionados ao número bits, n.

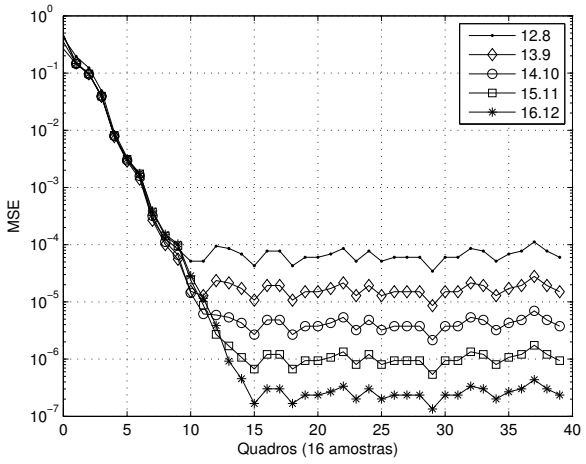

Figura 7. Resultados de desempenho da RBF para os testes realizados para o cenário da porta XOR.

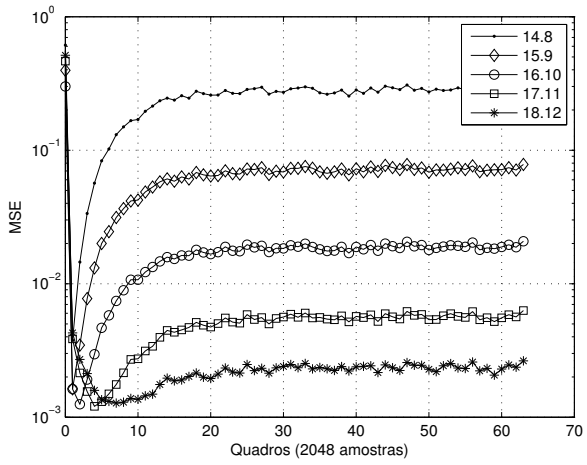

Figura 8. Resultados de desempenho da RBF para os testes realizados para o cenário com a função Seno.

V. Conclusões

Este trabalho apresentou uma proposta de implementação paralela em ponto fixo para FPGA de uma RBF treinada com o algoritmo do LMS. Todos detalhes de implementação da proposta foram apresentados e analisados em termos de área de ocupação, resolução em bits e atraso de processamento. A estrutura proposta foi submetida a testes para vários formatos de resolução em dois cenários amplamente conhecidos, caracterizados como o problema de classificação não linear com a porta XOR e o problema de interpolação utilizando a função Seno. Os resultados obtidos são bastante significativos e apontam para a possibilidade de utilização da proposta em situações práticas mais complexas.

Referências

[1] S. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., ser. Prentice Hall International Editions Series. Prentice Hall, 1999.

[2] A. Savich, M. Moussa, and S. Areibi, "The impact of arithmetic representation on implementing mlp-bp on fpgas: A study," *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 240–252, 2007.

[3] "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239 – 255, 2010.

[4] C. Latino, M. Moreno-Armendáriz, and M. Hagan, "Realizing general mlp networks with minimal fpga resources," in *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, 2009, pp. 1722–1729.

[5] S. Hariprasath and T. N. Prabakar, "Fpga implementation of multilayer feed forward neural network architecture using vhdl," in *Computing, Communication and Applications (ICCCA), 2012 International Conference on*, 2012, pp. 1–6.

[6] A. A. Hassan, A. Elnakib, and M. Abo-Elsoud, "Fpga-based neuro-architecture intrusion detection system," in *Computer Engineering Systems, 2008. ICCES 2008. International Conference on*, 2008, pp. 268–273.

[7] J. Patra, T. Devi, and P. Meher, "Radial basis function implementation of intelligent pressure sensor on field programmable gate array," in *Information, Communications Signal Processing, 2007 6th International Conference on*, 2007, pp. 1–5.

[8] S. Brassai, L. Bako, G. Pana, and S. Dan, "Neural control based on rbf network implemented on fpga," in *Optimization of Electrical and Electronic Equipment, 2008. OPTIM 2008. 11th International Conference on*, 2008, pp. 41–46.

[9] J.-S. Kim and S. Jung, "Evaluation of embedded rbf neural chip with back-propagation algorithm for pattern recognition tasks," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, 2008, pp. 1110–1115.

[10] Y.-S. Kung, M.-S. Wang, and T.-Y. Chuang, "Fpga-based self-tuning pid controller using rbf neural network and its application in x-y table," in *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on*, 2009, pp. 694–699.

[11] J. seob Kim and S. Jung, "Implementation of the rbf neural chip with the on-line learning back-propagation algorithm," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2008, pp. 377–383.

[12] P. Evert, R. Amudhan, and P. Paul, "Implementation of neural network based controller using verilog," in *Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on*, 2011, pp. 353–357.

[13] A. Gargouri, M. Krid, and D. Masmoudi, "Hardware implementation of pulse mode rbfnn based edge detection system on virtex v platform," in *Systems Signals and Devices (SSD), 2010 7th International Multi-Conference on*, 2010, pp. 1–5.

[14] I.-C. Vizitiu, I. Rîncu, A. Radu, I. Nicolaescu, and F. Popescu, "Optimal fpga implementation of garbf systems," in *Optimization of Electrical and Electronic Equipment (OPTIM), 2010 12th International Conference on*, 2010, pp. 774–779.

[15] J. S. Kim and S. Jung, "Joint control of roboker arm using a neural chip embedded on fpga," in *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on*, 2009, pp. 1007–1012.

[16] Xilinx. Xilinx System Generator User's Guide. [Online]. Available: www.xilinx.com

[17] S. S. Haykin, *Adaptive Filter Theory*, 3rd ed. Prentice Hall, 1996.

[18] S. Al-Kazzaz and R. Khalil, "Fpga implementation of artificial neurons: Comparison study," in *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, 2008, pp. 1–6.