# RBF NEURAL NETWORK WITH LINEARLY APPROXIMATED FUNCTIONS ON FPGA

**Rudolf Marek, Miroslav Skrbek**

Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Computer Science & Engineering
121 35 Prague, Karlovo namesti 13, Czech Republic

*marekr2@fel.cvut.cz (Rudolf Marek)*

**Abstract**

This article focuses on implementation of the Radial Basis Function (RBF) neural network by using linearly approximated functions. The presented approach is suitable for hardware implementations on FPGA that may accelerate the simulation of neural networks of this type. The approach of linearly approximated functions goes beyond the complexity of multipliers and large look-up tables for implementing activation functions and replaces them with simpler logic circuits based on adders and shifters. We show that some combination of building blocks consisting of many multiplexers (if implemented separately) can be replaced by a small set of invertors (if combined). This considerably reduces the amount of hardware necessary to implement the RBF neurons. Further, we present the results of our pilot implementation of the RBF neural network on FPGA consisting of arithmetic blocks for neural calculations, memory blocks for prototype storage, and controllers. Bus interconnections among neurons and command based control of neurons provide very good scalability of the proposed architecture. It allows increasing the number of RBF neurons as well as increasing the dimension of the input vector. The pipelining technique used for RBF neurons allows full utilization of the hardware. All functional blocks of the RBF neural network are implemented in synthesizable VHDL, simulated by the ModelSim 6.2d VHDL simulator and synthesized for the Xilinx Virtex-4 device by the Xilinx ISE 8.2i. As a result of synthesis, we provide a number of parameters such as the maximum clock frequency and the number of function blocks that characterize the resulting synthesized FPGA design.

**Keywords: neural network, Radial Basis Function (RBF), linear approximation, neural hardware, FPGA.**

**Presenting Author's biography**

Rudolf Marek is a postgraduate student and researcher at the Department of Computer Science and Engineering. He received his Master's degree in Electronics and Computer Science & Engineering from the Faculty of Electrical Engineering, Czech Technical University in Prague in 2006. His research is focused on hardware acceleration of computational intelligence algorithms.

*ISBN 978-3-901608-32-2*  

## 1. Introduction

Recent years have seen rapid growth in the importance of neural networks and other computational intelligence paradigms in many applications [3][4][5]. Radial Basis Function (RBF) [1] can learn much faster than other comparable neural paradigms, therefore, they are suitable for real-time data processing and real time control. Neural networks used for this kind of application require fast simulation platforms to achieve short input-to-output latencies and fast responses to outer events. One solution is hardware acceleration of the simulation process by implementing the neural networks completely or partially in hardware [6] [7] using FPGA as a rapid prototyping platform.

This article is based on our previous research presented in [2], where we introduced a set of linearly approximated functions suitable for the implementation of neural networks. The approach of linearly approximated functions goes beyond the complexity of multipliers and large look-up tables and replaces them with simpler logic circuits based on adders and shifters. Although our previous research introduces the key building blocks for the RBF neural networks, it is solely focused on perceptrons.

In this article we go farther by showing that some combination of building blocks consisting of many multiplexers (if implemented separately) can be replaced by a small set of invertors (if combined). This considerably reduces the amount of hardware necessary to implement the RBF neurons. On the base of these results we proposed new functional units suitable for efficient implementation of the RBF neural networks in hardware. The proposed units were verified and evaluated on our FPGA implementation.

In Section 2, we give a short introduction to RBF neural networks and their implementation by linearly approximated functions. Section 3 is an overview of linearly approximated functions relevant to the proposed functional units. Section 3.6 describes a model of the proposed RBF neuron including its detailed evaluation. Section 4 describes hardware implementation of RBF neural network and presents results obtained from a FPGA synthesis tool.

## 2. RBF Neural Network

A single output, Radial Basis Function (RBF) neural network [1] can be generally described by the equations Eq. (1) and Eq. (2) as follows

$$\varphi_k^2 = \sum_{\mathrm{i}=1}^{\mathrm{N}} (\mathrm{x_i} - c_{ki})^2 \,, \tag{1}$$

$$y = \sum_{k=1}^{M} w_k G(\varphi_k) \,, \tag{2}$$

where $x_i$ is the i-th input and $c_{ki}$ is the i-th centroid component of the k-th RBF neuron, N is the dimension of the input vector, and M is the number of RBF neurons. $G(\varphi_k)$ is typically the Gaussian function defined as follows

$$G(\varphi_k) = \mathrm{e}^{-\lambda_k \varphi_k^2} \,, \tag{3}$$

where $\lambda_k = \frac{1}{2\sigma_k^2}$ determines the slope of the $G(\varphi_k)$ function. We can rewrite equations Eq. (1) and Eq. (2) so that we replace multiplication by addition in the $w_k G(\varphi_k)$ expression. Then the output of the RBF network is equal to

$$y = \sum_{k=1}^{M} e^{-\lambda_k \varphi_k^2 + \log(w_k)} \,. \tag{4}$$

By replacing the $e^x$ and $ln(x)$ with corresponding functions with the base of two, we obtain

$$y = \sum_{k=1}^{M} 2^{-\lambda_k \varphi_k^2 + \log_2(w_k)} \,, \tag{5}$$

where $\lambda_k$ is considered as a power of two constant.

By applying the linearly approximated functions according to [2], we obtain

$$\varphi_k^2 = \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{SQR_2}(\mathrm{x_i} - c_{ki}) \,, \tag{6}$$

$$\xi_k = \mathrm{LOG_2}(\mathrm{AFR_2}(\lambda_k \varphi_k^2)) \,, \tag{7}$$

$$y = \sum_{k=1}^{M} \mathrm{EXP_2}(\xi_k + \mathrm{LOG_2}(w_k)) \,, \tag{8}$$

where $SQR_2$ is a linearly approximated $x^2$ function, $LOG_2$ is a linearly approximated $log_2(x)$ function, $EXP_2$ is a linearly approximated $2^x$ function and $AFR_2$ is a linearly approximated Gaussian-like function corresponding to the $2^{-x}$ function.

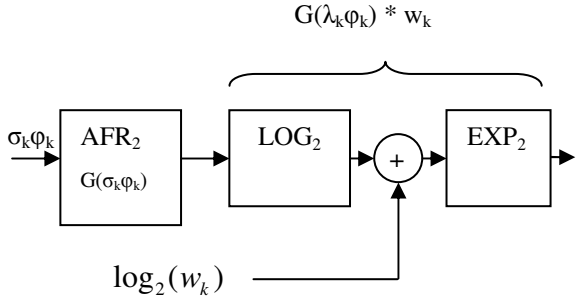


Figure 1: Multiplication of the RBF neuron output by weight $w_k$ composed from the $AFR_2$, $LOG_2$, and $EXP_2$ blocks.

 

It can be shown that $LOG_2(AFR_2(x))$ can be reduced to a simple logic inversion operation that replaces the chain of $AFR_2$ and $LOG_2$ blocks.

This allows rewriting Eq. (7) as follows

$$\xi = \mathrm{LOGAFR}_2(\lambda_k \varphi_k^2), \tag{9}$$

which corresponds to $\xi = -\lambda_k \varphi_k^2$ formula.

Since $\log_2(w_k)$ is a constant that can be entirely stored in a register, Eq. (8) can be implemented as shown in Figure 2. Compare the lower complexity with the corresponding circuit in Figure 1.

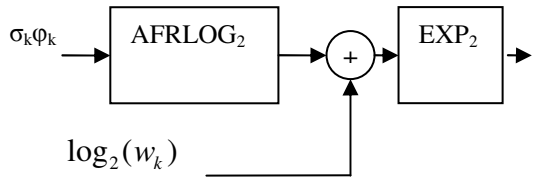


Figure 2: Multiplication of the RBF neuron output by weight $w_k$ composed from the $AFRLOG_2$ and $EXP_2$ blocks.

## 3. Linearly Approximated Functions

In this section, we give an overview of the set of linearly approximated functions that were published in [2] and are used in our RBF neural network implementation.

### 3.1. SQR

The $SQR_2$ function can be described by the equation

$$\mathrm{SQR}_2(x) = (\mathrm{a} + 2\mathrm{c}) \overset{0}{>>} (x_{n-1} - n) \tag{10}$$

where $x$ is in the range of $\langle 0,1)$, $a = 2^n$, n is the position of the left-most one in a binary representation of $x$, $b = x_{n-1} 2^{n-1}$, and $x_{n-1}$ is the value of the bit which is the right side neighbor of the left-most one. We can write that $c = \mathrm{x} - \mathrm{a} - \mathrm{b}$. Note that n is always negative for x in the range of $\langle 0,1)$. The $\overset{0}{>>}$ operator represents the shift-right operation inserting zeros from the left.

### 3.2. LOG

The linearly approximated $LOG_2$ function is defined as follows:

$$\mathrm{LOG}_2(x) = \mathrm{floor}(\log_2(x)) + \frac{\mathrm{x}}{2^{\mathrm{floor}(\log_2(x))}} - 1 \tag{11}$$

where $x$ is in the range of $\langle 0,1)$.

This function is realized by a shifter (set of multiplexers) that shifts the binary representation of x to the left until the left-most one of $x$ appears at the $2^0$ position. The integral part of the result is equal to 15-n, where n is the number of shifts. The fractional part is composed from $x$ shifted by $n$ to the left but with the left-most one excluded. For our 16-bit binary representation of $x$, the shifter has four stages shifting by 8, 4, 2, 1 to the left. The result is in sign-and-magnitude binary representation consisting of the sign bit, 5 bits of the integral part and 14 bits of the fractional part.

### 3.3. EXP

The linearly approximated $EXP_2$ function is defined as follows

$$\mathrm{EXP}_2(x) = 2^{\mathrm{int}(x)}(1 + \mathrm{frac}(x)), \tag{12}$$

where frac(x) is the fractional part of $x$. This function is implemented by a four-stage shifter to the right. The number of shifts is given by the integral part of $x$. The shifter shifts the fractional part of $x$, which must be extended by one from the left.

### 3.4. AFR

The $AFR_2$ function is a linearly approximated $2^{-x}$ function. This is described by the following equation

$$\mathrm{AFR}_2(x) = \frac{1}{2^{\mathrm{int}(|\mathrm{x}|)}} \left(1 - \frac{\mathrm{frac}(|x|)}{2}\right) \tag{13}$$

This function is implemented by a four-stage shifter that shifts the inverted value of one half of the fractional part of the absolute value of $x$.

### 3.5. AFRLOG

The proposed $AFRLOG_2$ function covers the functionality of both $LOG_2$ and $AFR_2$ blocks. From Eq. (11) and Eq. (13) it can be derived that

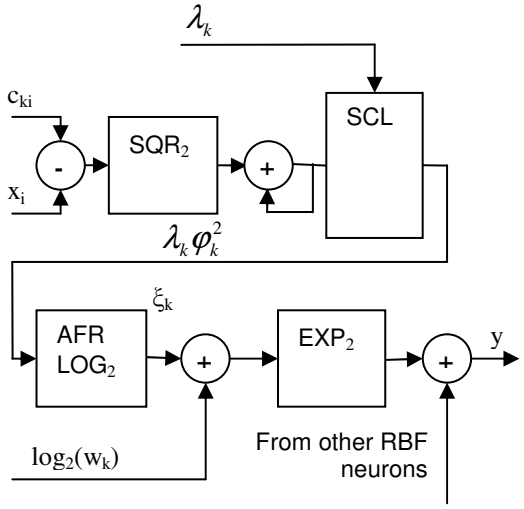


Figure 3: Model of the Proposed RBF Neural Network

 

$$\mathrm{AFRLOG}_2(|x|) = -\mathrm{int}(|x|) - \mathrm{frac}(|x|) \qquad (14)$$

The AFRLOG function is implemented as a set of invertors that corresponds to multiplication of $x$ by -1 for one's complement binary representation of $x$.

### 3.6. Model of the Proposed RBF Neural Network

A model of the proposed RBF neural network is shown in Figure 3. This model is composed of linearly approximated functions described in the previous section. The subtracter followed by SQR block and accumulator calculates the Euclidean distance of the input vector and centroid. The Euclidean distance is passed through the SCL block which shifts the value to the left or right according to required $\lambda_k$ which is limited to powers of two. The chain of the AFRLOG block followed by an adder and the EXP block calculates the activation function of the RBF neuron which is multiplied by the weight of the output neuron.

The proposed model was written in Java and evaluated from functionality and precision points of view. Figure 4 depicts the response of a two-dimensional RBF neuron with linearly approximated functions. The very low impact of linearly approximated functions on the RBF neuron behavior can be seen. The maximum absolute error produced by two input RBF neuron is equal to ±0,0736 (i.e. 7,36% of the output range).

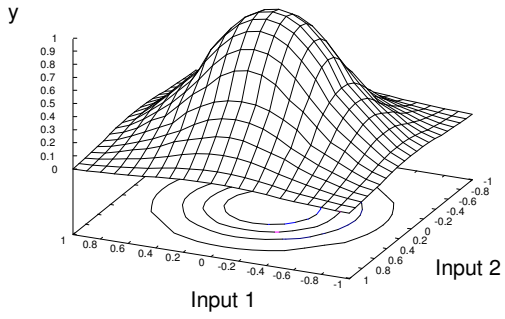


Figure 4: Response of the two-input RBF neuron implemented by linearly approximated functions. The y axis represents the output of the neuron.

## 4. Hardware implementation

The RBF neural network was implemented in synthesizable VHDL on the Xilinx Virtex 4 FPGA.

The main goal of the design was to exploit maximum parallelism for the given neural network topology allowing all neurons to work in parallel and produce the output just after processing all vector elements. The only part which cannot be truly parallelized is the final linear weighted sum of RBF neuron outputs. In the worst case scenario, it would take as many clocks as the number of RBF neurons in the design. Two approaches solving this problem were considered: a tree of adders and pipelining.

The tree of adders would require $\log_2(n)$ clocks to accomplish the operation, but also require $n$-1 functional units.

Without a tree of adders, the output neuron needs as many clocks as the number of RBF neurons, but we used pipelining to overlap the computation and summation. This allows the RBF neurons to compute a new vector while summing the previous output value. This means that full utilization of the hardware may be achieved if the number of RBF neurons is equal to the dimension of the input vector. In further text we denote such units as base units.

Larger networks may be built from multiple base units. In this case, the sum operation is implemented as a tree of adders.

### 4.1. Base unit

We created a base unit containing a single output RBF network with sixteen RBF neurons, one linear weighted output neuron and accepting sixteen dimensional input vectors. The RBF neurons in this unit take exactly the same number of clocks as the input dimension. The structure of the base unit is shown in Figure 5.

The dimension of the input vectors in our design was arbitrarily chosen. The design can be easily extended to support more dimensional input to fit specific needs.

The design goal of the proposed base unit was to ease its scalability in terms of neuron used per unit, as well as to ensure that even many of such units will be able to easily perform the computations together. Therefore, all neurons are interconnected with four different buses:

- command bus
- address bus
- input data bus
- inner data bus/output enable bus

All neurons are connected to the command and address buses. Each neuron has a unique address assigned and may respond to commands issued on the command bus only if its address matches the address issued on the address bus. There are also broadcast addresses that allow feeding of all neurons with the same command or data. This is suitable for fetching a vector component for all RBF neurons at once. The outputs of RBF neurons are connected to the inner data bus, which is connected to the output neuron (S).

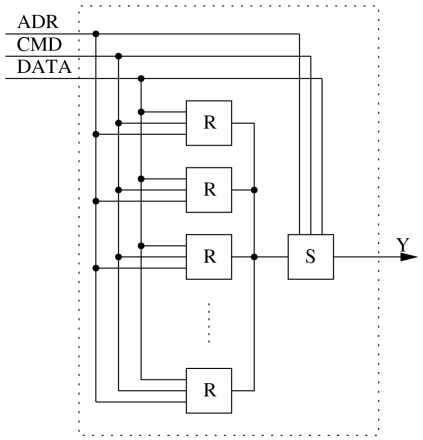


Figure 5: Structure of the Base Unit

Each RBF neuron accepts the following set of commands:

- compute the output based on input vector elements
- load the weights into the neuron
- load the scale factor into the neuron
- compute the weighted sum of RBF neurons output (for the output neuron)

Loading of the internal weight/scale register is achieved though the input data bus.

### 4.2. RBF Neuron Implementation

The RBF neuron implements equations Eq.(6), Eq.(7) and Eq. (8) according to Figure 3. The structure of the RBF neuron is shown in Figure 6.

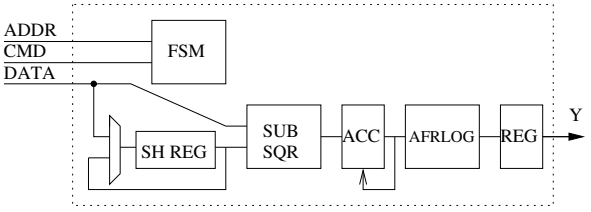


Figure 6: Structure of the RBF Neuron

All blocks inside the neuron are controlled by the finite state machine, which controls the computations based on commands fed from the command bus.

The SUB-SQR block and AFRLOG block are implemented as combinatorial logic circuits providing output within a single clock cycle. The Euclidean distance is accumulated in the ACC block, which is a sequential circuit with the need for clocking. For a 16-component vector, the accumulator requires 16 clock cycles to complete the computation. In each clock cycle, a new vector component is fed from the data bus. Simultaneously, in each clock cycle, individual centroid components are pulled from a circular shift register to the subtracter. Each neuron has its own shift register for storing components of the centroid. After processing of all vector components, the Euclidean distance is fed into the AFRLOG block. The output of the AFRLOG block is stored in a register, from which it is sent to the output neuron.

### 4.3. Output neuron implementation

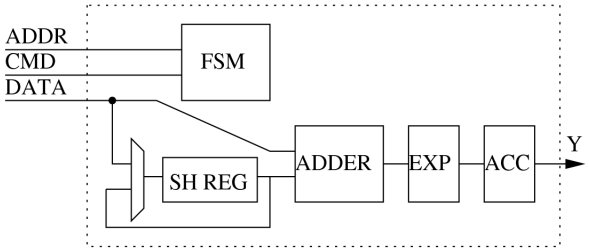


Figure 7: Structure of the Output Neuron

Figure 7 depicts the structure of the output neuron, which calculates the linear weighted sum.

The output neuron is designed similarly to the RBF neuron. The weights are stored in logarithmic form in the shift register. In each clock cycle, one of all RBF neuron outputs is enabled and added with corresponding weight together. The sum is passed into the EXP block (combinatorial circuit) and then stored in the accumulator. After all neurons have been processed, the accumulator contains the output of the RBF network.

### 4.4. Design testing and verification

The FPGA design was tested and verified successfully against the software model written in Java. VHDL test benches were generated using additional bash shell scripts and loaded into the VHDL simulator. Simulation of the VHDL code was performed in the ModelSim 6.2d simulator software.

### 4.5. Synthesis results

VHDL code synthesis was performed with Xilinx ISE 8.2i.

The base unit with 16 RBF neurons and one output neuron runs on a frequency up to 96MHz. The table below shows achieved speeds for a selected number of neurons connected to the unit.

| Number of RBF Neurons | Frequency [MHz] |
|---|---|
| 8 | 100.8 |
| 16 | 96.3 |

Table 1: Maximum Clock Frequency of the Base Unit

As shown in Table 1, the design scales well with a growing number of neurons connected to the base unit.

The design was optimized for speed and synthesized for the Xilinx Virtex-4 device (4vlx25sf363-12).

 

Table 2 summarizes the FPGA resources used for the base unit implementation with 16 RBF neurons. Macroblocks that were synthesized for the base unit are shown in Table 3.

| Resource name | Absolute number of FPGA resource | Total percentage of FPGA resource |
|---|---|---|
| Slices | 3775 | 35% |
| Slices Flip/Flop | 1104 | 5% |
| 4 input LUT used as logic | 7294 | 35% |
| 4 input LUT used as shift register | 274 | 0.1% |

Table 2: Resources of FPGA Occupied by the Base Unit

| Macro name | Count |
|---|---|
| 4-bit up counter | 17 |
| 16-bit register | 320 |
| 18-bit register | 32 |
| 4-bit register | 16 |
| 5-bit register | 16 |
| 8-bit comparator equal | 16 |
| 1-of-16 decoder | 1 |
| 18-bit tristate buffer | 16 |

Table 3: Macroblocks Synthesized for the Base Unit

## 5. Conclusion

In this paper we present fully parallel FPGA synthesizable VHDL implementation of the RBF network using linearly approximated functions. We showed that the combination of activation function and logarithm function results in a very simple operation (negation), which significantly simplifies the hardware implementation of the whole RBF neural network. The proposed RBF neural network consists of adders and multiplexers, no multiplier or large look-up tables are necessary.

Bus interconnections among neurons and command based control of neurons provide very good scalability of the proposed architecture. It allows increasing the number of RBF neurons as well as increasing the dimension of the input vector. The pipelining technique used for RBF neurons allows full utilization of the hardware.

We presented the synthesis results as well as the usage of RTL blocks obtained from the Xilinx ISE 8.2. The results show that more than 32 RBF neurons at a minimum can be implemented in 4vlx25sf363-12 FPGA. For a base unit containing 16 RBF neurons, we can achieve a processing rate equal to 1,5GCPS at 96MHz clock frequency. The architecture was simulated by the ModelSim VHDL simulator and validated against the model written in Java.

This research is supported by the research program "Transdisciplinary Research in the Area of Biomedical Engineering II" (MSM6840770012) sponsored by the Ministry of Education, Youth and Sports of the Czech Republic and the internal grant “Hardware Accelerated Computational Intelligence” of the Czech Technical University in Prague (CTU0707313).

## 6. References

[1] Haykin, S.: Neural Networks a Comprehensive Foundation. Macmillan College Publishing Company, New York, 1994, ISBN 0-02-352761-7

[2] Skrbek, M.: Fast Neural Network Implementation. In: *Neural Network World*. 9, No. 5, (1999) p. 375-391. ISSN 1210-0552.

[3] Zhi-gang, L., Jun-zheng, W: RBF-Neural Network Adaptive PID Control for 3-Axis Stabilized Tracking System. Proceedings of the Sixth International Conference on Hybrid Intelligent Systems (HIS'06), 2006

[4] Shen, M., Zhang, Y., Li, Z., Yang. J, Beadle, P.: Real-Time Detection of Signal in the Noise Based on the RBF Neural Network and Its Application. ISNN 2004, *LNCS 3174*, pp. 350–355, 2004.

[5] Chang, C., Fu, S.: Image Classification using a Module RBF Neural Network. Proceedings of the First International Conference on Innovative Computing, Information and Control (ICICIC'06), 2006.

[6] Krips, M., Lammert, T., and Kummert, A: FPGA Implementation of a Neural Network for a Real-Time Hand Tracking System. Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications, 2002.

[7] Pandya, V., Areibi, S., Moussa, M.: A Handel-C Implementation of the Back-Propagation Algorithm On Field Programmable Gate Arrays. Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs, 2005.

[8] *Virtex-4 User Guide*. Technical document Xilinx Inc., April, 2007. http://www.xilinx.com.

 