High precision FPGA implementation of neural network activation functions

Francisco Ortega-Zamorano, José M. Jerez, Gustavo Juárez, Jorge O. Pérez, Leonardo Franco

Abstract— **The efficient implementation of artificial neural networks in FPGA boards requires tackling several issues that strongly affect the final result. One of these issues is the computation of the neuron's activation function. In this work, a detailed analysis of the FPGA implementations of the Sigmoid and Exponential functions is carried out, in a approach combining a lookup table with a linear interpolation procedure. Further, to optimize board resources utilization, a time division multiplexing of the multiplier attached to the neurons was used. The results are evaluated in terms of the absolute and relative errors obtained and also through measuring a quality factor and the resource utilization, showing a clear improvement in relationship to previously published works.**

I. Introduction

FPGAs [1] are reprogrammable silicon chips, using prebuilt logic blocks and programmable routing resources. They can be configured to implement custom hardware functionality, and in this sense, FPGAs are completely reconfigurable and can almost instantly change its behavior by recompiling a new circuitry configuration. In recent years, the advance in technology made possible to construct FPGAs with considerable large amounts of processing power and memory storage, permitting their application in several areas such as Telecommunications, Robotics, Pattern recognition tasks, Infrastructure monitoring, etc. [2]. As FPGAs are intrinsically parallel devices, they are quite suitable for Neural Network implementations, and so several studies have analyzed their application [3], [4], [5]. A broad classification of FPGA neural network applications can be done according to whether they include the learning process ("on-chip implementations") [6], [5] or if the training of the neural network model is performed externally in a Personal Computer (PC) where the FPGA acts as a hardware accelerator ("off-chip implementations") [7], [8]. Programming a FPGA is not a trivial task as they are predominantly codified using hardware description languages such as VHDL or Verilog, languages that are complex making the programming process very time consuming in most cases. An important aspect at the time of the implementation of an algorithm in a FPGA regards the data type representation. The nature of the FPGAs encourages the use of a fixed point representation because this type of representation is more efficient. A floating point representation might be used but this would require the utilization of specific cores [9], [10]. Regarding this issue, the work of Savich et al. (2007) [11] describes an interesting analysis of the implementation of floating point neural algorithms in fixed point arithmetic.

In the present work, we analyze the implementation in a FPGA board of Sigmoid and Exponential functions, two functions related to artificial neural network implementations. The Sigmoid function (σ) is one of the most used activation function in neural networks [12] , and as such its implementation in FPGA has been analyzed by several authors in the past [13], [14], [4]. The Exponential function is less commonly used in neural network models but it is considered in this work as it is related to the Upstart and C-Mantec constructive neural network algorithms that constitute a valid alternative to traditional backpropagation trained neural networks [16].

In the present work, we develop a method based on using a lookup table approach combined with a linear interpolation scheme. Similar approaches have not been used much in the past due to the memory requirements for the storage of table values, but given the actual specifications of FPGA boards it is a very interesting possibility [13], that may lead to fast and accurate results.

II. Methods

We analyze in this work the implementation on a FPGA board of two mathematical functions involved in the implementation of neural network models: the Sigmoid and Exponential functions. The board used for the current implementation is the Virtex-5 OpenSPARC Evaluation Platform (ML509). This device includes a Xilinx Virtex-5 XC5VLX110T FPGA that provides different connector devices that includes 2 USB ports, 2 PS/2 ports, RJ-45 (10/100/1000 Networking) and RS-232 connectors. The VIRTEX 5 board used has 69120 programmable LUTs (Look-Up Tables), 148 RAM/FIFO available blocks of memory and 64 DSP48 modules. Each DSP48 slice contains a 25 x 18 multiplier, an adder, and an accumulator, containing extensive cascade capabilities to efficiently implement high-speed DSP algorithms.

The Sigmoid function (σ) is one of the most commonly used activation function in neural networks [12] and is defined as:

$$\sigma(x) = \frac{1}{1 + exp(-x)}. \quad (1)$$

The Exponential function is a basic matematical function usually used to model a quantity that grows or decays at

F. Ortega-Zamorano, J.M Jerez and L. Franco are with the Department of Computer Science, Malaga University, Spain (email: {lfranco,jja,ortega}@lcc.uma.es), and J. Pérez and G. Juárez are with the Department of Computer Science, Tucumán National University, Argentina (email: {jperez,gjuarez }@herrera.unt.edu.ar)

This work was supported by grants from Junta de Andalucía P10-TIC-5770 and P08-TIC-04026, and from CICYT (Spain) through grant TIN2010-16556 (all including FEDER funds).

978-1-4799-4485-9/14/$31.00 ©2014 IEEE

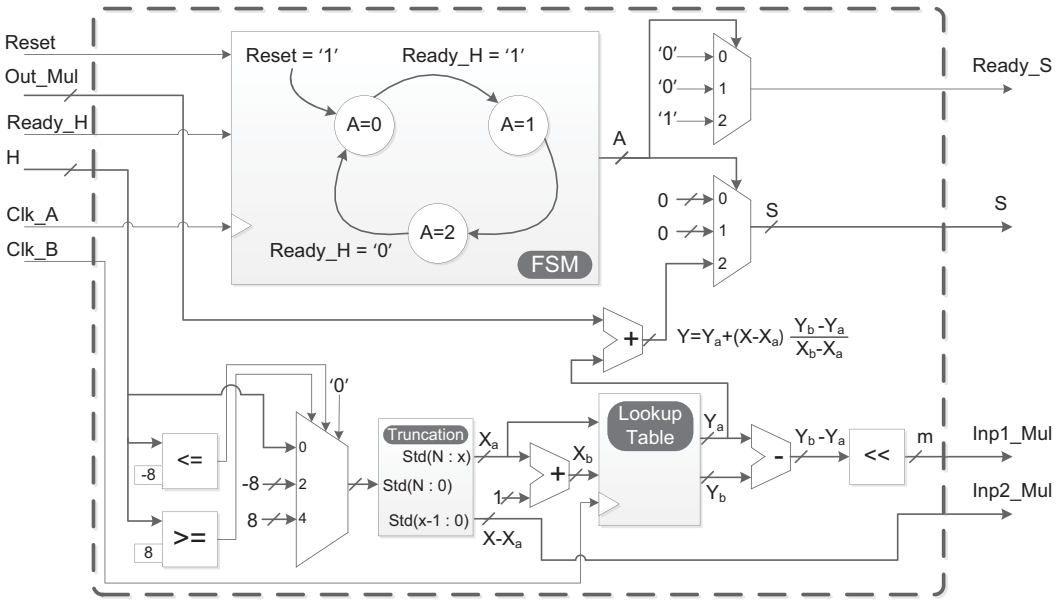

Fig. 1. Hardware representation of the interpolation block used for the computation of the functions.

a rate proportional to its current value, and it is used in several neural network algorithms like the Upstart [15]), and C-Mantec Constructive Neural Network algorithm [16].

The implementation of the Exponential and Sigmoid functions was carried out using a lookup table approach combined with a linear interpolation procedure. The lookup table contains equispaced values of the function in a range of input values of interest and the linear interpolation procedure uses two adjacent tabulated values (lower and larger) with respect to the input in order to compute the approximation, according to the following equation:

$$Y = Y_a + (X - X_a)\frac{(Y_b - Y_a)}{(X_b - X_a)}, \tag{2}$$

where X_a and X_b are the closest lower and upper values in relationship to the input one (X) that are stored in a lookup table, while Y_a and Y_b are the corresponding values of the function being computed.

The implementation has been carried out using fixed point arithmetic, as this type of representation is most suitable when using FPGAs [17].

Figure 1 shows hardware representation of the interpolation block. The inputs for the computation of the function are indicated on the upper left corner of the diagram: From top to bottom we have first a Reset signal that initialize the finite state machine (FSM), the multiplier output (Out_Mul), a signal (Ready_H) to indicate that the value of H is the correct one to be used, H the argument of the Sigmoid function (also known as the synaptic potential of the neuron), and two clock signals, Clk_A the system frequency and Clk_B the double of Clk_A. Within the figure, the upper part shows the finite state machine that controls the timing of the process, while in the lower part of the figure it is shown the procedure in charge of computing the approximation of the function, first by checking the closest values stored in the lookup table, and secondly by the computation of the linear interpolation (Eq. 2). On the right side of the diagram the outputs of the procedure are shown: Ready_S is a signal to indicate that the value of the function S is available, and near the bottom left part Inp1_Mul and Inp2_Mul are the inputs to a multiplier that will return Out_Mul. When the FSM is in the state $A = 1$ Inp1_Mul contains the value of $m = \frac{(Y_b - Y_a)}{(X_b - X_a)}$ while Inp2_Mul is equal to $X - X_A$. These two values are sent to a multiplier and the result is returned to the block as Out_Mul, so the value of S can be returned when the FSM is in state A=2 according to Eq. 2. As it can be appreciated in the FSM the whole procedure can be executed in two system clock cycles (from A=0 to A=2), because in this last state S is ready to be used in further computations.

The resources employed for the implementation of the interpolation process are shown in Table I as a function of the length of the representation used, formed by the integer part (N_a) and the decimal part (N_b). (Note that the values shown in the table do not take into account resources involved in the construction of the lookup table that are considered separately). Regarding the operation frequency of the block, in the worst case (N_a=N_b=16) the interpolation block generates a delay of 1.152 ns that implies a frequency of 868.056 MHz, higher than the maximum allowed frequency, so the system operating frequency will be determined by the rest of the hardware resources.

A. Multiplier

The strategy for the implementation of both functions was to use a time-division multiplexing strategy for the multiplier module permitting to compute sequentially the values and saving board resources. Figure 2 shows a schematic representation of a neuron in the FPGA in which the time-division

TABLE I

EMPLOYED RESOURCES IN THE IMPLEMENTATION OF THE INTERPOLATION PROCESS USED FOR THE COMPUTATION OF THE SIGMOID AND EXPONENTIAL FUNCTIONS.

N_a	N_b	LUTs	Registers
8	8	43	26
8	12	55	30
8	16	72	34
12	12	60	30
12	16	75	34
16	16	78	34

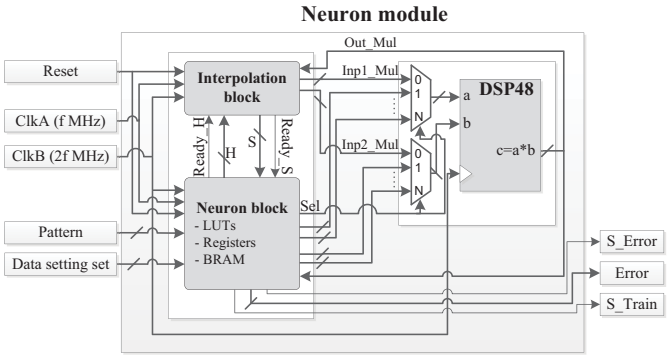

Fig. 2. Scheme of an implemented neuron that uses a time-division multiplexed strategy to execute several multiplications without adding complexity to the interpolation process.

TABLE II

FPGA MEMORY REQUIREMENTS IN LUTS FOR DIFFERENT VALUES OF N_L AND N_T.

N_T \ N_L	16	32	64	128	256	512	1024
8	8	12	16	30	32	210	388
12	12	18	24	46	52	434	796
16	20	26	32	62	84	490	924

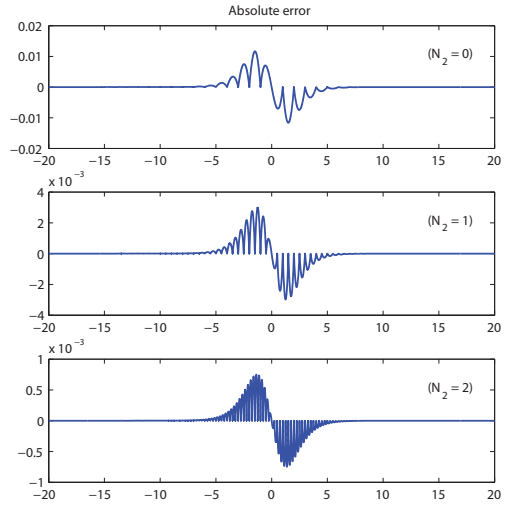

Fig. 3. Results obtained for the implementation of the Sigmoid functions for different values of N_2.

multiplexing of the DSP multiplier has been implemented. For an optimal utilization of the resources, the multiplier of each neuron has been performed with a synchronous DSP48 with an operation frequency two times faster than the one of the finite state machine, in order to perform the multiplication operation in one clock cycle.

B. Lookup Table

The computation of the sigmoid function utilizes a lookup table for storing tabulated values to be used in the interpolation procedure. The size of the table (N_L) is determined by the precision (number of bits) of the inputs. These values (X_a and X_b in Eq. 2) have been represented using a fixed point scheme with N_1 bits for the integer part and N_2 for the decimal part. In this way the total number of values in the table is defined by $N_L = 2^{N1+N2}$ for functions with only positive inputs or $N_L = 2^{1+N1+N2}$ for the case of functions with negative and positive inputs like the sigmoid function. To determine adequate values of N_1 and N_2, first, the value of N_2 is increased until the absolute error of the output function is lower than the desired value, to then apply the same process for N_1. To finish, the number of bits for representing the tabulated values (N_T) has to be determined. Table II shows the number of used LUTs for the different representation for the implementation of a lookup table, depending on $N_T = \{8, 12, 16\}$, and $N_L = \{16, 32, 64, 128, 256, 512, 1024\}$. A lookup table with any configuration of N_T and N_L can also be implemented by one block RAM instead of using LUTs, thus allowing for two different implementation possibilities that can be chosen according to the resource needs of the entire system.

III. RESULTS

A. Sigmoid function approximation

The approximation results obtained for the Sigmoid function are shown in Figs. 3, 4, and 5 where the absolute errors are shown as a function of the input value. Fig. 3 shows the result for three different values of $N_2 = \{0, 1, 2\}$, while Fig. 4 shows the result for three different values of $N_1 = \{1, 2, 3\}$. Fig. 5 shows the final approximation results as a function of $N_T = \{8, 12, 16\}$.

From an analysis of the results shown in Figs. 3, 4, and 5, our choice for the implementation of the Sigmoid function is $N_1 = 3$, $N_2 = 2$, and $N_T = 16$, as this choice permits to obtain absolute errors below 10^{-3} that guarantees a correct operation of most artificial neural networks algorithm. Table III shows maximum error (Max) and Root Mean Square Error (RMSE) for different values of N_1 and N_2. The first row of the table shows the results obtained for the chosen set of values ($N_1 = 3$, $N_2 = 2$) while for the rest of the rows larger values of N_1 and N_2 are used for comparison.

B. Comparison with previous approaches

We compare below the results obtained in the the current approach with two previous works where different aspects are taken into account in the analysis. Comparisons with other existing approaches can not be performed because the

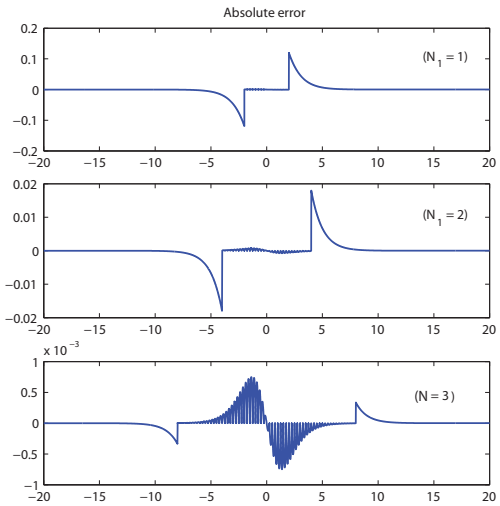

Fig. 4. Results obtained for the implementation of the Sigmoid functions for different values of N_1.

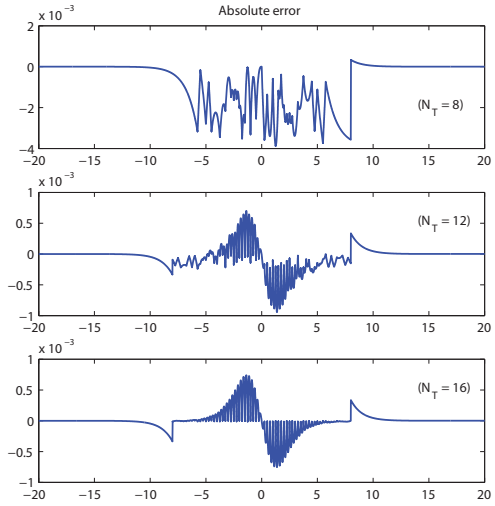

Fig. 5. Results obtained for the implementation of the Sigmoid functions for different values of N_T.

implementation details necessary for a reliable comparison were not given.

1) Comparison using a quality factor: In the work by Tommiska et al., 2003 [13] a quality factor Q has been introduced to analyze the accuracy and usability of a function implemented in a FPGA board:

$$Q = \frac{f_{max}}{LEs \cdot E_{ave} \cdot E_{max}}, \tag{3}$$

where f_{max} = clock rate, LEs = number of logic elements, E_{ave} = average error in per cent, and E_{max} = maximum error in per cent.

The quality factor (Q) for the Sigmoid approximation has been computed in two different cases, first by using the lookup table approximation alone (Q_1) and second when the interpolation scheme is used in combination with the lookup table (Q_2). The values of Q_1 and Q_2 are reported in Table IV together with previous published results [13]. For computing

TABLE III

MAXIMUM ERROR (MAX) AND ROOT MEAN SQUARE ERROR (RMSE) FOR DIFFERENT VALUES OF N_1 AND N_2 IN THE IMPLEMENTATION OF THE SIGMOID FUNCTION.

N_1	N_2	Max	RMSE
3	**2**	$7.548 \cdot 10^{-4}$	$2.447 \cdot 10^{-4}$
3	3	$3.353 \cdot 10^{-4}$	$9.477 \cdot 10^{-5}$
4	3	$1.982 \cdot 10^{-4}$	$4.428 \cdot 10^{-5}$
4	4	$6.091 \cdot 10^{-5}$	$1.394 \cdot 10^{-5}$
4	5	$2.669 \cdot 10^{-5}$	$8.783 \cdot 10^{-6}$
4	6	$1.755 \cdot 10^{-5}$	$8.362 \cdot 10^{-6}$

TABLE IV

QUALITY FACTOR FOR THE SIGMOID FUNCTION APPROXIMATIONS

Approximation	Q
Lookup table + interpolation	Q_2= 491.46
$sig_3 37p$ [13]	25.61
$sig_2 36p$ [13]	12.30
PLAN approximation [18]	1.743
Approximation of Alippi and StortiGajani[14]	1.085
Lookup table	Q_1 = 0.644
Approximation of Zhang [19].	0.227
A-law based approximation[20]	0.134

Q_1 the following values were used $f_{max} = 100$MHz, LEs= 32, $E_{ave} = 6.22$, $E_{max} = 0.78$, while for Q_2 the arguments were: $f_{max} = 100$MHz, LEs= $32 + 78 = 110$ (32 LEs for representing the table values, and 78 LEs for the interpolation procedure), $E_{ave} = 0.0755$, and $E_{max} = 0.0245$.

2) Comparsion based on resource utilization: A scheme of the Sigmoid function based on Taylor's theorem and Lagrange forms was proposed in [21], where a maximum allowable error(ϵ) of 0.01 was permitted in the hardware implementation. The necessary resources were one DSP and 11 LUTs, with the circuit performing the computation in 7 clock cycles at a maximum frequency of 373.5 MHz. The present work leads to a maximum allowable error of 0.001, using a DSP but in a multiplexing scheme involving no extra resources. Regarding the number of LUTS, the current implementation needs a larger resource utilization (up to 78 LUTs) because the circuit performs the computation in only 2 clock cycles with a maximum operation frequency of 868.056 MHz.

C. Exponential function approximation

The results for the exponential function approximation are shown in Figs. 6,7, and 8. Fig. 6 shows the result for three different values of $N_2 = \{0, 1, 2, 3\}$, while Fig. 7 shows the result for three different values of $N_1 = \{1, 2, 3\}$. Fig. 8 shows the final approximation results as a function of $N_T = \{8, 12, 16\}$.

From the observed results presented in Figs. 6,7, and 8 the choice made for the implementation of the Sigmoid function is $N_1 = 3$, $N_2 = 3$, and $N_T = 16$, as these values are the lowest ones permitting to obtain absolute errors below 2×10^{-3}.

Table V shows maximum error (Max) and Root Mean Square Error (RMSE) for different values of N_1 and N_2.

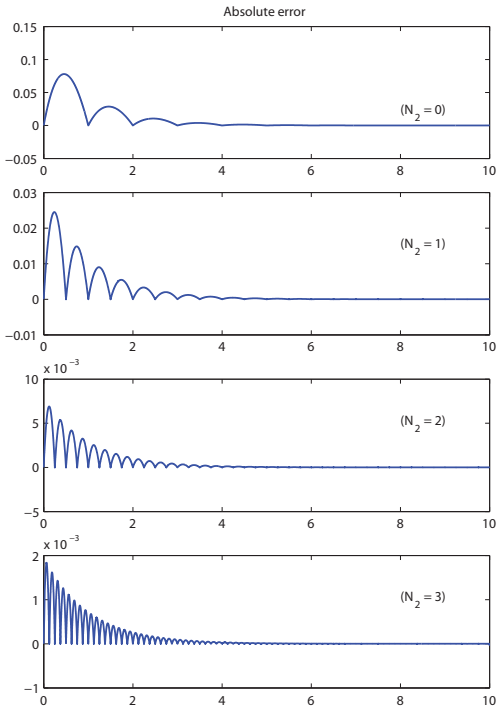

Fig. 6. Results obtained for the implementation of the exponential functions for different values of N_2.

TABLE V

MAXIMUM ERROR (MAX) AND ROOT MEAN SQUARE ERROR (RMSE) FOR DIFFERENT VALUES OF N_1 AND N_2 IN THE IMPLEMENTATION OF THE EXPONENTIAL FUNCTION.

N_1	N_2	Max	RMSE
3	**3**	$1.833 \cdot 10^{-3}$	$3.249 \cdot 10^{-4}$
3	4	$4.704 \cdot 10^{-4}$	$7.632 \cdot 10^{-5}$
4	4	$4.704 \cdot 10^{-4}$	$5.501 \cdot 10^{-5}$
4	5	$1.151 \cdot 10^{-4}$	$1.358 \cdot 10^{-5}$
4	6	$2.530 \cdot 10^{-5}$	$6.493 \cdot 10^{-6}$

The first row of the table shows the results obtained for the chosen set of values ($N_1 = 3$, $N_2 = 3$) while for the rest of the rows larger values of N_1 and N_2 are used for comparison.

In order to analyze the efficiency of the approximation done we have computed absolute and relative errors for the Exponential function in an arbitrarily selected interval between two tabulated values. Table VI displays the results for input values in the interval [2.000 2.125] showing in the columns the input values, the value of the exponential function with 6 significant digits, the value obtained with the approximation made and the absolute and relative errors. The extreme values of the interval considered corresponds to tabulated values used to compute the interpolation and so they appear in the table in boldface font.

IV. CONCLUSIONS

We have analyzed in this work the FPGA implementation of two basic functions related to the functioning of neural network algorithms by using a time division multiplexing strategy for carrying the multiplication operations in combination with a lookup table and interpolation scheme. A quality comparison done with respect to previously published results [13] for the Sigmoid function shows that the current implementation is very efficient, as we obtained a quality factor 40 times larger than the best previous published value, noting that this value is obtained without a significant increase of resource utilization. Regarding the lookup table plus interpolation scheme, it is also possible to conclude that the interpolation procedure helps very much to improve the approximation results, as the quality factor without the interpolation scheme reduces from 491 to 0.64. A further comparison to the results published in [21] shows that the current implementation is much faster, performing the computation of the Sigmoid function in only 2 clock cycles instead of the 7 used in the mentioned work.

Regarding the approximation of the Exponential function, the results can be considered of similar quality to those

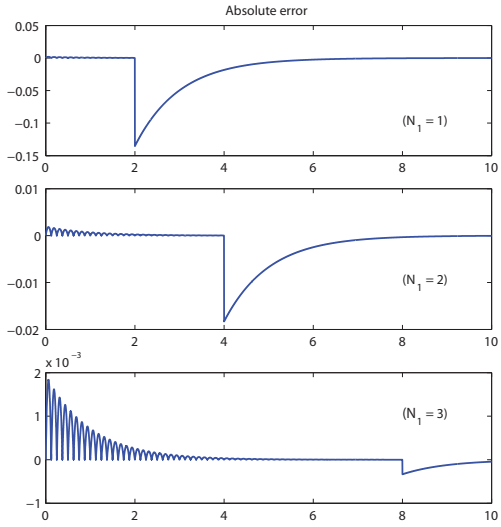

Fig. 7. Results obtained for the implementation of the exponential functions for different values of N_1.

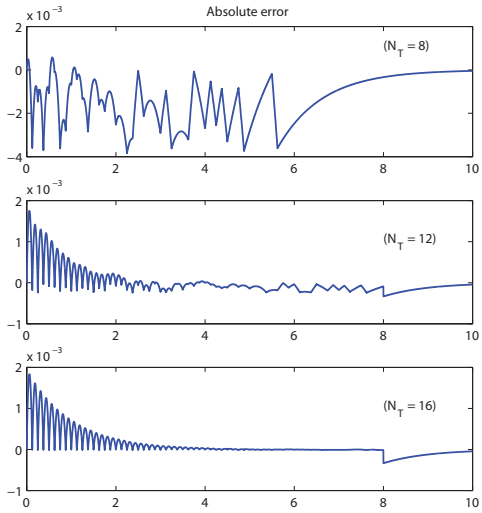

Fig. 8. Results obtained for the implementation of the exponential functions for different values of N_T.

TABLE VI

ABSOLUTE AND RELATIVE ERRORS FOR THE EXPONENTIAL FUNCTION FOR INPUT VALUES IN THE INTERVAL [2.000 2.125] FOR THE APPROXIMATION MADE USING A LOOKUP TABLE COMBINED WITH A LINEAR INTERPOLATION.

x	$exp(-x)$	Approximation	Absolute error	Relative error
2.000	**0.135335**	0.135330	$5 \cdot 10^{-6}$	$4 \cdot 10^{-5}$
2.025	0.131994	0.132150	$1 \cdot 10^{-4}$	$1 \cdot 10^{-3}$
2.050	0.128735	0.128970	$2 \cdot 10^{-4}$	$2 \cdot 10^{-3}$
2.075	0.125556	0.125790	$2 \cdot 10^{-4}$	$2 \cdot 10^{-3}$
2.100	0.122456	0.122610	$1 \cdot 10^{-4}$	$1 \cdot 10^{-3}$
2.125	**0.119433**	0.119430	$3 \cdot 10^{-6}$	$3 \cdot 10^{-5}$

obtained for the Sigmoidal function, even if the quality factor has not been compared as there were no previous published results. The results shown in table VI confirm the accuracy of the approximation, as very low error values were obtained. It is worth noting that the proposed techniques used for the functions approximation were obtained maximizing the utilization of FPGA resources, so that they can be replicated several times in the same board allowing for the construction of larger networks of neurons.

REFERENCES

[1] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Wiley-IEEE Press, 2007.

[2] E. Monmasson, L. Idkhajine, M. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "Fpgas in industrial control applications." *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, pp. 224–243, 2011.

[3] J. Zhu and P. Sutton, "Fpga implementations of neural networks - a survey of a decade of progress," in *Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL 2003*. Springer-Verlag, 2003, pp. 1062–1066.

[4] A. Gomperts, A. Ukil, and F. Zurfluh, "Development and implementation of parameterized fpga-based general purpose neural networks for online applications." *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 78–89, 2011.

[5] A. Omondi and J. Rajapakse, *FPGA Implementations of Neural Networks*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[6] K. Lakshmi and M. Subadra, "A survey on fpga based mlp realization for on-chip learning," in *International Journal of Scientific & Engineering Research*, vol. 4, 2013.

[7] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in fpga using layer multiplexing for effective resource utilization," *IEEE Transactions on Neural Networks*, pp. 880–888, 2007.

[8] S. Jung and S. S. Kim, "Hardware implementation of a real-time neural network controller with a dsp and an fpga for nonlinear systems," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, pp. 265–271, 2007.

[9] C. H. Ho, C. W. Yu, P. Leong, W. Luk, and S. J. E. Wilton, "Floating-point fpga: Architecture and modeling." *IEEE Trans. VLSI Syst.*, vol. 17, no. 12, pp. 1709–1718, 2009.

[10] Z. Jovanovic and V. Milutinovic, "Fpga accelerator for floating-point matrix multiplication," *IET Computers & Digital Techniques*, vol. 6, no. 4, pp. 249+, 2012.

[11] A. W. Savich, M. Moussa, and S. Areibi, "The impact of arithmetic representation on implementing mlp-bp on fpgas: A study," *IEEE Transactions on Neural Networks*, vol. 18, pp. 240–252, 2007.

[12] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall, 1994.

[13] M. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proc. Comput. Digit. Techn.*, vol. 150, no. 6, pp. 403–411, 2003.

[14] C. Alippi and G. Storti Gajani, "Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning," in *Proc. IEEE Int. Symp. on Circuits and Systems, Singapore*. IEEE Press, 1991, pp. 1505–1508.

[15] M. Frean, "The upstart algorithm: a method for constructing and training feedforward neural networks," *Neural Computation*, vol. 2, no. 2, pp. 198–209, Apr. 1990.

[16] J. Subirats, L. Franco, and J. Jerez, "C-mantec: A novel constructive neural network algorithm incorporating competition between neurons," *Neural Networks*, vol. 26, pp. 130–140, 2012.

[17] P. P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. John Wiley & Sons, 2008.

[18] H. Amin, K. Curtis, and B. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," *IEE Proceedings - Circuits, Devices and Systems*, vol. 144, pp. 313–317(4), December 1997.

[19] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations." *IEEE Trans. Computers*, vol. 45, no. 9, pp. 1045–1049, 1996.

[20] D. Myers and R. Hutchinson, "Efficient implementation of piecewise linear activation function for digital vlsi neural network," *Electron. Lett.*, no. 25, pp. 1662–1663, 1989.

[21] I. Campo, R. Finker, J. Echanobe, and K. Basterretxea, "Controlled accuracy approximation of sigmoid function for efficient fpga-based implementation of artificial neurons," *Electronics Letters*, vol. 49, no. 25, pp. 1598–1600, 2013.