Sigmoid Function Implementation Using the Unequal Segmentation of Differential Lookup Table and Second Order Nonlinear Function

Syahrulanuar Ngah and Rohani Abu Bakar
Faculty of Computer Systems & Software Engineering, UMP.
syahrulanuar@ump.edu.my

***Abstract*—This paper discusses the artificial neural network (ANN) implementation into a field programmable gate array (FPGA). One of the most difficult problem encounters is the complex equation of the activation function namely sigmoid function. The sigmoid function is used as learning function to train the neural network while its derivative is used as a network activation function for specifying the point at which the network should switch to a true state. In order to overcome this problem, two-steps approach which combined the unequal segmentation of the differential look-up table (USdLUT) and the second order nonlinear function (SONF) is proposed. Based on the analysis done, the deviation achieved using the proposed method is 95%. The result obtained is much better than the previous implementation that uses equal segmentation of differential look-up table.**

***Index Terms*—Differential Look-Up Table; FPGA; Second Order Nonlinear Function; Sigmoid Function.**

I. INTRODUCTION

Artificial neural networks (ANN) is an information processing system that aims to simulate human brain's architecture and function. It is now a popular subject in many fields and widely used in many applications such as speed estimation [1], pattern recognition and classification [2], control application, function approximation, optimization [3][4][5] and also as embedded system [6]. Almost any problem with high complexity can be solved by multilayer perceptron of ANN [1]. One of the advantages of ANN is the ability of parallel processing that makes it a useful computational tool in practice [7]. Even without the inner working knowledge of neural network elements itself, the designer can apply the ANN [5][8]. A clear disadvantage of the software implementation with ANN is a slow execution for the real-time applications [7] and in fact, it is not achievable when the several stages of code are needed to be executed sequentially [8]-[10].

Recently, hardware implementation has become important due to the performance gains of hardware systems compared to software implementation [11],[12]. When implementing the ANN into hardware, certain measures are to be taken to minimize the hardware usage since the hardware has limited of memory. Essentially, there are two types of hardware solutions. Complementary metal-oxide semiconductor (CMOS) device is an analogue hardware is one of the solutions. However analogue hardware undergoes inexact computation result and lack re-programmability [8]. Though the field programmable gate array (FPGA) digital hardware solution is the most interesting implementation of ANN, after taking into consideration of higher processing speed, cost of each implementation, reliability, flexible and reprogrammable architecture [1][13][14]. The challenge of this approach lies in how to implement the neuronal activation function when involving the complex equation with limited hardware resources to achieve the high-precision of ANN output.

II. ACTIVATION FUNCTION

ANN consists of a huge class of different architecture. Several factors need to be considered when implementing the ANN to solve a certain problem since the ANN only perform excellently when the selection is matched perfectly with the targeted problem. Multiple feedforward networks, one of the important class of ANN consist of the input layer, hidden layer and output layer. Each neuron from the previous layer feeds every neuron on the next layer. Neuron accumulates the sum of each it is input. The output value of each neuron is determined by the activation function of each neuron. Figure 1 shows the general structure of multilayer artificial neural network.

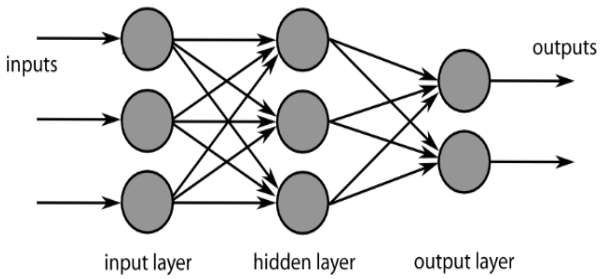

Figure 1: General structure of multilayer artificial neural network.

Figure 2 shows one of the neurons where the calculation of the activation function involved. Basically, there are many types of activation function such as hard-limiter, piecewise linear, hyperbolic tangent and sigmoid function. Figure 3 showing the pattern of the hard limiter and piecewise linear activation function.

Sigmoid function and hyperbolic tangent function represented by Equation (1) and (2) respectively, are most commonly used as an activation function [15][16][17].

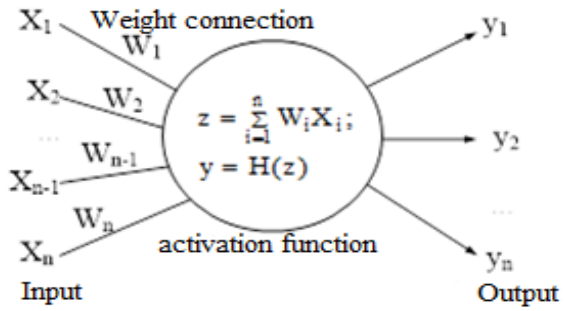

Figure 2: Abstracted model of neuron with connection.

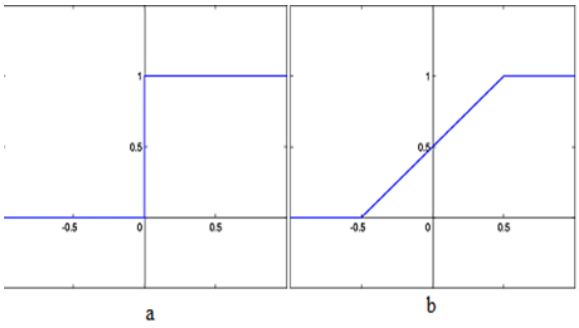

Figure 3: (a) Hard Limiter and (b) Piecewise Linear activation function

$$f(x) = \frac{1}{1+e^{-x}} \quad (1)$$

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (2)$$

However, there is lack of theoretical background for the selection of the activation function [18]. Shamseldin et. al. [19] and Shrestha et. al. [15] indicated that transfer function is interchangeable as long as they are in sigmoid shape. The sigmoid function is the activation function focused in this study since it is the most frequently used in back propagation neural network [18][20].

The sigmoid function is used as learning function for training the neural network while it derivative is used as a network activation function for specifying the point at which the network should switch to a true state. Transition in which improves the neural response; unlike the hard-limiter or saturated linear activation function. [4]. Figure 4 shows the curve of the sigmoid function.

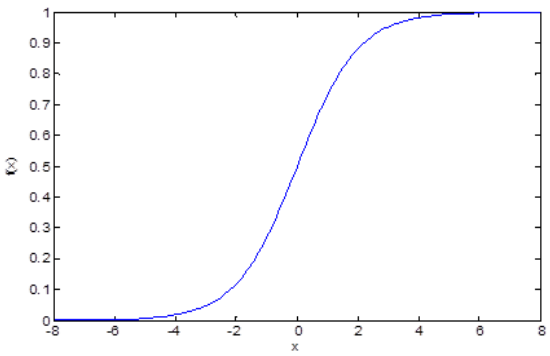

Figure 4: Sigmoid function curve

In the hardware implementation of ANN, the computation of a sigmoid function is one of the factors that constrain either the computation time or occupied area of the system. Currently, a few methods have been proposed and three of the following approaches are widely being used for realizing the sigmoid function into FPGA; lookup table (LUT), the piecewise linear approximation (PWL) and functional approximation in different input/output interval. LUT-based evaluation is the fastest among these three methods [21]. There are also other methods such as coordinate rotation digital computer (CORDIC) function [22], piecewise linear approximation for nonlinear (PLAN) function [23] and second order nonlinear function (SONF) [24].

A. Lookup Table (LUT)

The simplest implementation of the sigmoid function is using LUT. With LUT, the function is approximated by a limited number of uniformly distributed point [25]. The sigmoid function curve will be uniformly divided into segments as shown in Figure 5.

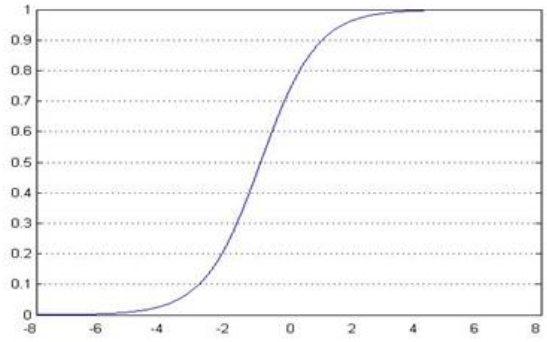

Figure 5: LUT implementation

The value of each segment is stored in a table. This method presents the fastest design since it involves a delay of only one memory-access time to produce the result [3][21]. However, to achieve a higher degree of accuracy, the area requirement increases exponentially. The deviation achieved by this method is ranging from -0.005 to 0.005 with the used of 16 Kb of hardware memory [8]. However, the LUT design does not optimize well under floating point format. To some extent, the inbuilt RAM available in FPGA is used to realize the sigmoid function to optimize the area. Although this technique is simple to implement, when higher precision is needed, it requires a large area of hardware [2][26]. Hence it would be impractical to implement the LUT in a massive parallel ANN.

B. Piecewise Linear Approximation (PWL)

The piecewise approximation method approximates by dividing the sigmoid function into five linear segments called pieces. Equation (3), representing the segments of the sigmoid function. The accuracy of the approximation can be achieved by increasing the number of segments, subsequently increasing the area utilization of hardware also. Based on this method, H. Amin et. al. [23], proposed an efficient piecewise linear approximation of a nonlinear function (PLAN). Table 1, shows the implementation of PLAN technique to approximate the sigmoid function.

$$f(x)=\begin{cases}0, if\ \ x\ \le -8,\\ (8)-|x|\frac{1}{64}, if\ \ -8<x\ \le\ 1.6\\ \frac{x}{4}+0.5,\ \ if\ \ |x|<1.6\\ 1\text{-}(8\text{-}|x|)\frac{1}{64}, if\ \ 1.6\ \le\ x<8\\ 1\ if\ \ x>8\end{cases} \quad (3)$$

Table 1
Implementation of PLAN to Approximate the Sigmoid Function

Operation	Condition
Y=1	|x| >= 5
Y=0.03125 * |x| + 0.84375	2.375 =< |x| < 5
Y=0.125 * |x| + 0.625	1 =< |x| < 2.375
Y=0.25 * |x| + 0.25	0=< |x| < 1
Y = 1 - Y	X < 0

The significance of this method is that the multiplications operation can be replaced by simple shift operations; for example, if X = 2, then X is shifted 3 times to the right (0.12510 = 0.001,) and then added to 0.625 (0.101,), and for X = -2 the same result is subtracted from 1. These shifts and add operations, however, can be totally removed and replaced with a simple logic design by performing a direct transformation from input to sigmoidal output [23]. Since this method replaced the need of multiply/add operation by a simple gate design, which leads to a very small and fast digital approximation of the sigmoid function. Figure 6 and Figure 7 shows the deviation achieved by using the PLAN function compared to the sigmoid function, and range of deviation between PLAN and sigmoid function respectively.

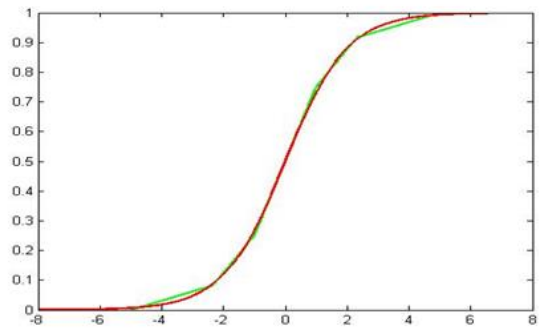

Figure 6: Output comparison between sigmoid function and PLAN

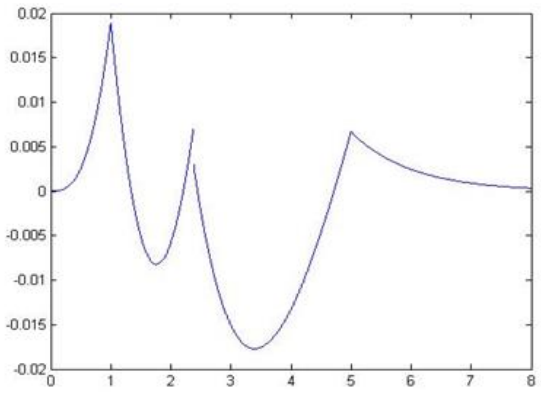

Figure 7: Deviation range between sigmoid function and PLAN

C. Piecewise Nonlinear Approximation

Another method that can be used to implement the sigmoid function is piecewise second order approximation. Generally, this method approximated the sigmoid function by:

$$f(x)=c_0+c_1*x+c_2*x^2 \quad (4)$$

The main drawback of this method is the need of two multiplication and two additions, which result in a low conversion speed [3][8]. Then, Zhang et. al. [24], has presented second order nonlinear function (SONF). Zhang has divided the sigmoid function into 4 segments, represented by:

$$f(x)=\begin{cases}1-\frac{1}{2}(1-\frac{1}{4}|x|)^2, 0<x\le 4\\ \frac{1}{2}(1-\frac{1}{4}|x|)^2,\ -4<x\ \le\ 0\\ 1\ , 4<x\\ 0,\ x\le\ -4\end{cases} \quad (5)$$

After simplification, this method can be implemented with one multiplier, two shifters and two XORs. Figure 8 shows the comparison output between the sigmoid function and SONF. Meanwhile, Figure 9 shows the plotted graph of deviation between sigmoid function and SONF where the deviation ranging from -0.022 to 0.022 [8]. However, this method lacks accuracy compared to LUT or CORDIC function.

D. CORDIC Function

The trigonometric CORDIC algorithms were originally developed as a digital solution for real-time navigation problems. The original work is credited to Jack Volder [27]-[29]. This method is an efficient algorithm for computing the elementary function such as hyperbolic function, multiplication and division. CORDIC function gains accuracy at the cost of latency, where latency is defined as the number of clock cycles required from the start of the calculation until the resulting data is ready. In other words, more iteration is needed to achieve higher accuracy. In order to get the deviation ranging from -0.005 to 0.005, the CORDIC required 50 clock cycle to achieve that [22].

E. Hybrid Method

All the method discussed above have their own advantages and disadvantages. Though, the demand for higher accuracy with using fewer hardware resources and less computation time still there. To get optimum balance between accuracy and the hardware memory usage, the researcher now starts doing the hybrid method by combining two or more methods together for realizing the sigmoid function into FPGA. Ngah et. al. have proposed the combination of SONF and differential lookup table (dLUT) [30]. Basically, the ideas of Ngah et. al. paper is used the deviation value between the SONF and sigmoid function as shown in Figure 9 to create another LUT namely differential lookup table (dLUT). The deviation value is then divided equally into 64 segments and stored into dLUT. That's mean each segment will have the same value.

By using this two-steps approaches, the deviation of the sigmoid function can be reduced. The deviation is ranging from -0.0022 to 0.0022. Figure 10 shows the achieved

Table 2
Summary of Previous Sigmoid Function Implementation

Implementation	Clock Cycle	Range different with Equation (1)	Memory used
LUT[22]	3	-0.005 to 0.005	16Kbits
CORDIC[22]	50	-0.005 to 0.005	0
SONF[24]	10	-0.022 – 0.022	0
Two-step implementation with equal segmentation[30]	13	-0.0022 to 0.0022	320 Bits

deviation comparison between the hybrid methods and the SONF.

Table 2 shows the summary of the previous implementation of the sigmoid function in hardware. It shows that the deviation achieved by the hybrid method 10 times better than the SONF and twice compared to LUT and CORDIC function methods.

III. Proposed Approach

Based on the idea proposed by Ngah et. al., this paper proposed improvement method on implementing the dLUT. The same two-steps are still used in this study. Figure 9 shows that the graph is symmetrical at X= 0 and Y= 0 (0,0). Therefore, the values needed to be stored in the dLUT are reduced in half.

Also, as can be seen in Figure 8 and Figure 9, the deviation for certain area are different. Along the X axis, the deviation from 0 to 1.2, and 3.6 to 8 are less compared to the deviation achieved from 1.2 to 3.6. Based on this situation, the value needed to be store for creating the differential LUT should be different. Instead of dividing the deviation value between SONF and sigmoid function, equally into 64 segments, this paper proposed the deviation value is divided unequally into 64 segments.

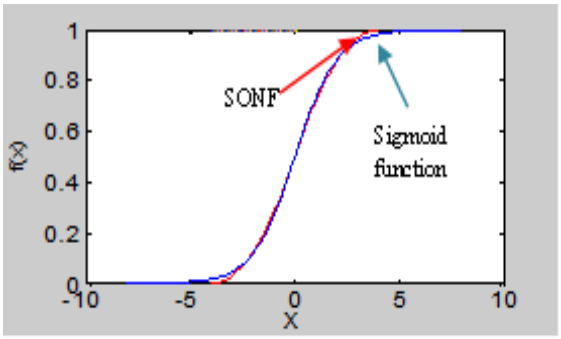

Figure 8: Output comparison between sigmoid function and SONF

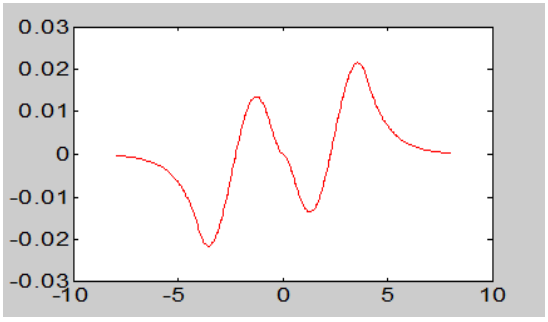

Figure 9: Deviation range between sigmoid function and SONF

To realize this idea, first the deviation value between SONF and sigmoid function are first divided into 3 main areas A, B and C. These areas then are divided into 12, 32 and 20 segments respectively. These numbers are choosing based on the deviation where area A is between -0.14 to 0, area B is between -0.014 to 0.022 and area C is between 0 to 0.022. These 3 areas (A, B and C) are having 3 different values. These values then are used to create dLUT. Figure 11 shows the proposed segmentation portion. Once the unequal segmentation of dLUT is created, it will be used to calculate the sigmoid function.

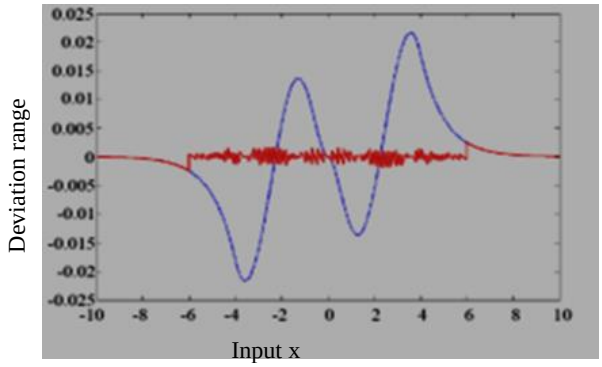

Figure 10: Deviation achieved by Ngah et. al. is lesser than the SONF

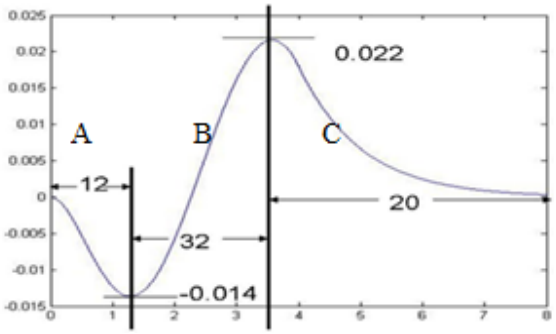

Figure 11: Deviation value between SONF and sigmoid function unequally divided into 64 segments

IV. Result And Discussion

Figure 12 shows the step of calculating the sigmoid function. First, the SONF is used to calculate the sigmoid function. As proposed, the deviation between the SONF and sigmoid function are used to create the dLUT by using unequal segmentation method. That's mean, the value in each segment in dLUT is already known either positive of negative. In the Second step, the output of SONF in the first step then is add/minus with the value from the dLUT to produce the final output of the sigmoid function. If the output from the first step is a positive number, and the value in the dLUT is a negative number, then these two number will be added. However, if the output from the first step in a positive number, and the value in the dLUT also a positive number, then the minus operation is used. By using this two-step approach, the combination of the SONF and the unequal segmentation of dLUT, the output of the sigmoid function can be reduced.

The simulation was running on Altera Cyclone IV DE-115 board. The program was implemented in Verilog HDL and post-simulated in Quartus II 13.0. The result from the simulation shows that the total of clock cycle and memory

needed are same as Ngah et. al. These results are as expected since this paper used the same method as Ngah et. al. On the other hand, by using the proposed idea, unequal segmentation of deviation between SONF and the sigmoid function to create the dLUT while maintaining the step used by Ngah et. al., the deviation value for implementing the sigmoid function in hardware is improved. The deviation achieved by using this method is ranging from -0.0006 to 0.0006. More than 95% improved compared to Ngah et. al. method.

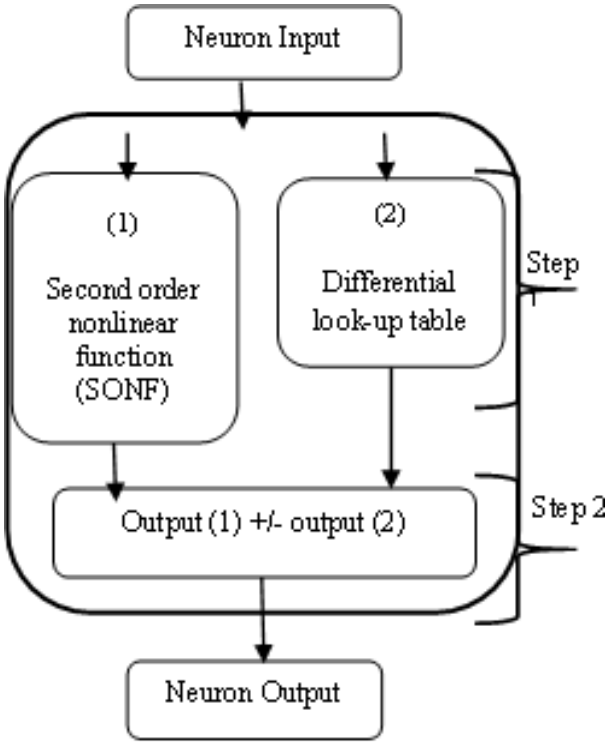

Figure 12: Two-steps implementation of the sigmoid function [30]

V. CONCLUSION

One of the characteristic features of ANN is performing the parallel mathematical calculations. Yet, this property is not achievable with PC software even with the fastest sequential processor when involving a large number of neurons where several stages of code are executed sequentially [7][9][31]. Implementing the ANN into the FPGA is one of the solutions. Challenge faced by the researcher is how to implement the activation function, especially the sigmoid function that involving the complex equation as shown by Equation (1). A few methods have been discussed. LUT is the simplest and fastest but required a huge amount of hardware resources to achieved higher accuracy. The same argument goes with CORDIC function. However, instead of hardware resources, CORDIC function suffers from long computation time since it required more iteration to achieve higher accuracy. Meanwhile, PWL and piecewise nonlinear approximation remove the need for multiplication and addition operation. Both methods need fewer hardware resources and are faster in computation time. However, in term of accuracy, both are still low compared to LUT and CORDIC. Then Ngah et. al. proposed two-steps approach where it had achieved 50% improvement compared to LUT and CORDIC function. While retain used the two-steps approach and proposed a new method by unequal segmentation for implementing the dLUT, this paper further reduces the deviation value of the sigmoid function by more than 95% compared to its predecessor.

REFERENCES

[1] H. M. Hasanien, "FPGA implementation of adaptive ANN controller for speed regulation of permanent magnet stepper motor drives," *Energy Convers. Manag.*, vol. 52, no. 2, pp. 1252–1257, Feb. 2011.

[2] V. Saichand, N. D. M., A. S., and N. Mohankumar, "FPGA Realization of Activation Function for Artificial Neural Networks," *2008 Eighth Int. Conf. Intell. Syst. Des. Appl.*, pp. 159–164, Nov. 2008.

[3] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi, "Efficient hardware implementation of the hyperbolic tangent sigmoid function," *2009 IEEE Int. Symp. Circuits Syst.*, pp. 2117–2120, May 2009.

[4] X. Chen, G. Wang, W. Zhou, S. Chang, and S. Sun, "Efficient Sigmoid Function for Neural Networks Based FPGA Design," in *International Conference on Intelligent Computing, ICIC*, 2006, pp. 672–677.

[5] E. Z. Mohammed and H. K. Ali, "Hardware Implementation of Artificial Neural Network Using Field Programmable Gate Array," *Int. J. Comput. Theory Eng.*, vol. 5, no. 5, pp. 780–783, 2013.

[6] A. Benjemmaa, I. Klabi, M. S. Masmoudi, J. El Ouni, and M. Masmoudi, "Implementations approaches of neural networks lane following system," in *Electrotechnical Conference (MELECON), 2012 16th IEEE Mediterranean*, 2012, no. 3, pp. 515–518.

[7] T. Orlowska-Kowalska and M. Kaminski, "FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system," *IEEE Trans. Ind. Informatics*, vol. 7, no. 3, pp. 436–445, 2011.

[8] M. T. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proc. - Comput. Digit. Tech.*, vol. 150, no. 6, p. 403, 2003.

[9] B. M. Wilamowski, "Neural network architectures and learning algorithms - How to not get frustrated with neural networks," *Industrial Electronics Magazine, IEEE*, vol. 3, no. 4, pp. 56–63, 2009.

[10] J. L. J. Liu and D. L. D. Liang, "A Survey of FPGA-Based Hardware Implementation of ANNs," *2005 Int. Conf. Neural Networks Brain*, vol. 2, pp. 915–918, 2005.

[11] A. Muthuramalingam, S. Himavathi, and E. Srinivasan, "Neural Network Implementation Using FPGA : Issues and Application," *Int. J. Inf. Technol.*, vol. 4, no. 2, pp. 396–402, 2008.

[12] M. Krips, T. Lammert, and a. Kummert, "FPGA implementation of a neural network for a real-time hand tracking system," *Proc. First IEEE Int. Work. Electron. Des. Test Appl. '2002*, pp. 313–317, 2002.

[13] Z. Xie, "A non-linear approximation of the sigmoid function based on FPGA," in *2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*, 2012, pp. 221–223.

[14] Y. Ago, Y. Ito, and K. Nakano, "An FPGA implementation for neural networks with the FDFM processor core approach," *Int. J. Parallel, Emergent Distrib. Syst.*, vol. Volume 28, no. No. 4, pp. 308 – 320, 2012.

[15] R. R. Shrestha, S. Theobald, and F. Nestmann, "Simulation of flood flow in a river system using artificial neural networks," *Hydrol. Earth Syst. Sci.*, vol. 9, no. 4, pp. 313–321, 2005.

[16] D. Costarelli and R. Spigler, "Approximation results for neural network operators activated by sigmoidal functions.," *Neural networks J. Int. Neural Netw. Soc.*, vol. 44, pp. 101–106, Aug. 2013.

[17] A. Mohamed, A. Issam, B. Mohamed, and B. Abdellatif, "Parallel and Mixed Hardware Implementation of Artificial Neuron Network on the FPGA Platform," *Int. J. Eng. Technol.*, vol. 6, no. 5, pp. 2008–2016, 2014.

[18] H. Yonaba, F. Anctil, and V. Fortin, "Comparing Sigmoid Transfer Functions for Neural Network Multistep Ahead Streamflow Forecasting," *J. Hydrol. Eng.*, vol. 15, no. 4, pp. 275–283, 2010.

[19] A. Y. Shamseldin, A. E. Nasr, and K. M. O. Connor, "Comparison of different forms of the Multi-layer Feed-Forward Neural Network method used for river flow forecasting," *Hydrol. Earth Syst. Sci.*, vol. 6, no. 4, pp. 671–684, 2002.

[20] T. M. Jamel and B. M. Khammas, "Implementation of a Sigmoid Activation Function For Neural Network Using FPGA," in *13th Scientific Conference of Al-Ma'moon University College*, 2012, no. April.

[21] P. K. Meher, "An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks," in *Proceedings of the 2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip, VLSI-SoC 2010*, 2010, pp. 91–95.

[22] R. W. Duren, R. J. Marks, P. D. Reynolds, and M. L. Trumbo, "Real-

time neural network inversion on the SRC-6e reconfigurable computer," *IEEE Trans. Neural Networks*, vol. 18, no. 3, pp. 889–901, 2007.

[23] H. Amin, K. M. Curtis, and B. R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," in *Circuits, Devices and Systems, IEE Proceedings* -, 1997, vol. 144, no. 6, pp. 313–317.

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

[25] K. Leboeuf, A. H. Namin, R. Muscedere, H. Wu, and M. Ahmadi, "High Speed VLSI implementation of the hyperbolic tangent sigmoid function," *2008 Third Int. Conf. Converg. Hybrid Inf. Technol.*, pp. 1070–1073, Nov. 2008.

[26] C. Lin and J. Wang, "A digital circuit design of hyperbolic tangent sigmoid function for neural networks," *2008 IEEE Int. Symp. Circuits Syst.*, pp. 856–859, May 2008.

[27] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *Electron. Comput. IRE Trans.*, vol. EC-8, no. 3, pp. 330–334, 1959.

[28] E. Deprettere, P. Dewilde, and R. Udo, "Pipelined cordic architectures for fast VLSI filtering and array processing," *ICASSP '84. IEEE Int. Conf. Acoust. Speech, Signal Process.*, vol. 9, pp. 250 – 253, 1984.

[29] Y. H. Hu and S. Naganathan, "Novel implementation of a chirp Z-transform using a CORDIC processor," *IEEE Trans. Acoust.*, vol. 38, no. 2, pp. 352–354, 1990.

[30] S. Ngah, R. A. Bakar, A. Embong, and S. Razali, "Two-Steps Implementation of Sigmoid Function for Artificial Neural Network in Field Programmable Gate Array," *ARPN J. Eng. Appl. Sci.*, 2015.

[31] I. Aybay, "Classification of neural network hardware," *Neural Netw. World, IDG Co.*, vol. 6, no. 1, pp. 11 – 29, 1996.