Parallel and Mixed Hardware Implementation of Artificial Neuron Network on the FPGA Platform

ATIBI Mohamed [#1], ATOUF Issam[#2], BOUSSAA Mohamed [#3,] BENNIS Abdellatif [#4]

[#] Laboratory of information processing, Hassan II – Mohammedia –Casablanca University
Cdt Driss El Harti, BP 7955 Sidi Othman Casablanca, 20702, Maroc
[1] atibi.simo@gmail.com
[2] issamatouf@yahoo.fr
[3] md.boussaa@gmail.com
[4] al_bennis@yahoo.fr

Abstract— Most applications in different fields (automotive, robotics, medical…) take advantage of the proven performance by artificial neural networks to solve their most complex problems. The architecture chosen for implementation is the multilayer perceptron that uses retro propagation as a learning algorithm. This article presents modular hardware implementation of multilayer perceptron architecture of artificial neuron network 'ANN', in the FPGA platform according to two models (parallel and mixed hardware implementation), and the comparison between these two implementations in terms of hardware resources and execution time. The two implementations are based on the proposed module of a formal neuron with the sigmoid activation function.

Keyword-ANN, hardware resource, execution time, FPGA, module of a formal neuron.

I. Introduction

The neurons are nerve cells that have the ability to process information in the human brain. They are interconnected and they operate in parallel in order to realize, with great speed, tasks of perception that are not within the reach of powerful modern computers.

These properties have contributed to the appearance of artificial neural networks, which are currently used in a wide variety of applications in science and engineering, such as classification and pattern recognition [1][2]. The ANN is consisting of layers of neurons that are responsible for the classification of data or learning. Compared to other popular machine learning algorithms, such as Bayesian network and ″support vector machine″ (SVM),the ANN presents numerous advantages, such as simplicity of structure, the parallelism of calculation, amelioration of classification performance and the adaptive learning parameters[3] [4].

The most used architecture of artificial neuron network is the multilayer perceptron (MLP) proposed by Gibson and al. [5]. The MLP is a complete connection structure, which uses sigmoidal functions such as activation of neurons. Its features are based on the ability to learn and solve complex problems, as optimization, classification and pattern recognition [6]. There are several learning algorithms; the most used with this type of architecture is the algorithm of retro propagation of errors (Back pro) which simulates the learning phenomenon by the error in the presence of a supervisor. This is an optimization algorithm which aims to determine the best weight possible connection.

However, the properties of artificial neuron network require an efficient hardware implementation circuit [7], hence the use of a platform capable of implementing these ANNs and use the parallelism in processing and reconfiguration [8].

Recently, the field-programmable gate array (FPGA)have been used in many applications requiring the implementation of ANNs, because the programmable logic offers more flexibility, speed and space to the implementation of logical elements[9]. It also enables the design of hardware architectures which synthesize the great neural networks consisting of hundreds of neuron units [10]. Thus, several attempts are reported in the literature that deals with the design of application--Specific Integrated Circuit (ASIC) comprising a plurality of processing units in parallel. However, related to principles` design of ASICs, the networks obtained were limited by the size and type of algorithm implemented [1].

Currently, the field-programmable gate arrays (FPGA) are the preferred reconfigurable hardware platform.

The current FPGAs offering performance and integration density of logic elements similar to ASICs with, too, Flexibility of design cycles / test faster. Also we find them frequently used in the fields of research and in industrial applications [1].

This article discusses in detail the hardware implementation of the Multilayer Perceptron including all neurons that have a sigmoid activation function, in the FPGA platform.

The article also discusses two types of implementations: The first is a parallel implementation and the second is considered as a mixed implementation because it contains a series and parallel implementation at the same time. These two implementations use megafunctions designed by the builders of the FPGA platform, manipulating real data in a floating point format with 32 bits (norm IEEE-754).

The article is structured as follows: After an introduction we begin Section II by defining some notion on artificial neural networks; Section III presents the implementation details of the two architectures (parallel and mixed), Section IV shows the results of tests performed during the implementation, and finally a conclusion of this article.

II. THEORY OF NEURAL NETWORKS

The artificial neural networks are mathematical models used to model two performances of the human brain: learning from a database of examples and generalization on examples not seen in the learning phase.

The artificial neuron network is a set of formal neurons (W.MCCulocch etW.Pitts) interconnected to achieve a well-defined task; each formal neuron is considered an elementary processor which receives at its input a number of parameters coming either upstream neurons or from the sensor component of the application device realized. These inputs are connected to the neuron by weights representing the strength of the connection called synaptic weight. This elementary processor has a single output, which in its turn, powers downstream neurons. Indeed the neuron performs a number of mathematical operations on its inputs: Multiplying each entry by its synaptic weight, summing the weighted inputs, and transfer this sum to the output through a function called an activation function or transfer function (e.g. Fig. 1).

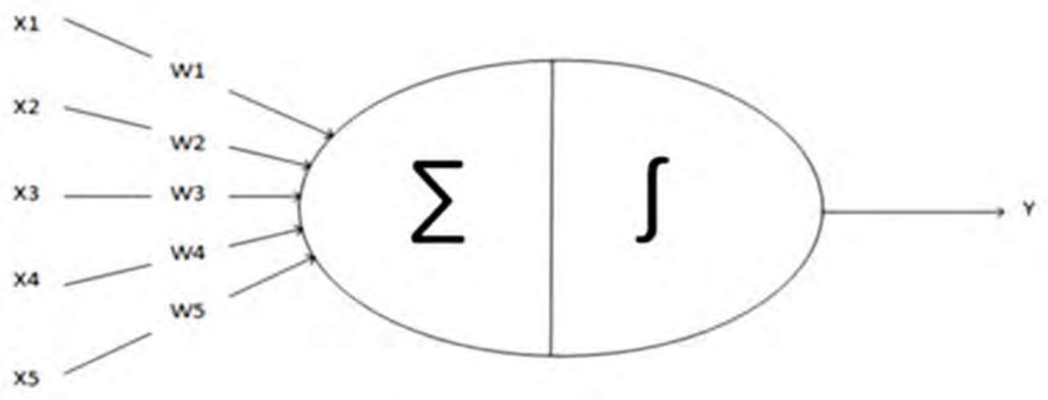

Fig 1. Schema of a formal neuron

The construction of RNA is firstly opting for the choice of architecture, namely, the manner in which these neurons are to be interconnected. Among the most architecture used in the literature, we find multilayer perceptron (MLP); this architecture is composed of an input layer, one or more hidden layers and an output layer (e.g. Fig. 2 [1]).

The input layer receives the input parameters for the task to accomplish. The hidden layers are used to determine the nonlinear frontiers and the output layer produces the result of calculation [11].

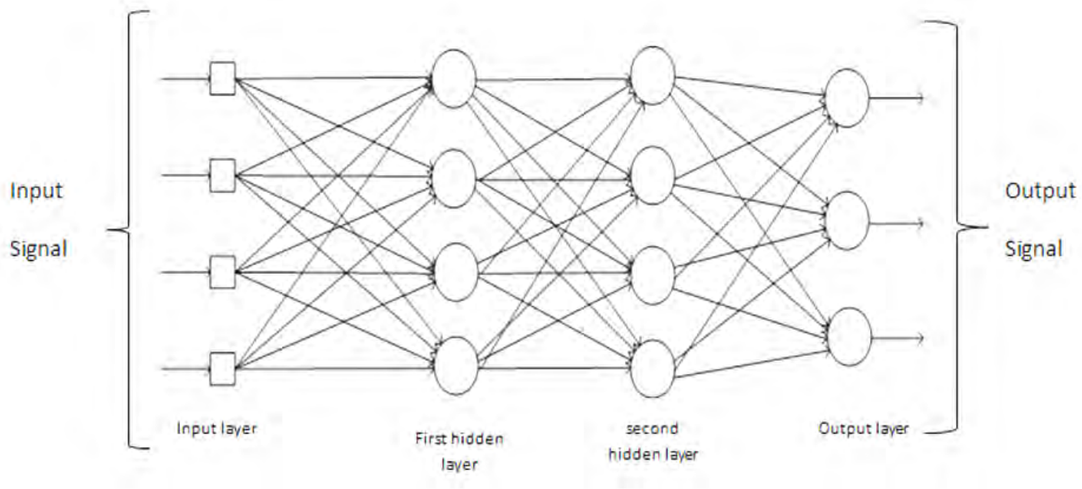

Fig 2. Schema of a multilayer perceptron

Secondly, using an adequate algorithm to adjust the synaptic weights of different connections. There are several learning algorithms; the most used with MLP is the algorithm of error back propagation (Back pro) which simulates the learning phenomenon by the error.

The use of this algorithm is to provide initially random values to synaptic weight, then present successively the elements of the training set to the network input and be evaluated for each element to the basis example the observed error in the output of each neuron; thereafter, exploit the value of this error to adjust the synaptic weights, the objective is to minimize the overall error using the method of gradient descent.

The application of this algorithm requires a number of steps:

- Calculation of the observed output:

After choosing architecture as example (e.g. Fig. 2), synaptic weight of the entire network is initialized with random values, and then the neuron of the input layer receives the parameter values of each element of the example base and transmits them to the neurons of the first hidden layer. In this layer each neuron calculates its own output according to equation (1):

$$Y_j = f(\sum_{i=1}^{n} W_{ji}.X_i) \quad (1)$$

The Y_j outputs will be used later as inputs to the next layer and this process will continue until the output layer whose neurons will calculate their observed outputs according to equation (2):

$$Y_k = f(\sum_{j=1}^{n} W_{kj}.Y_j) \quad (2)$$

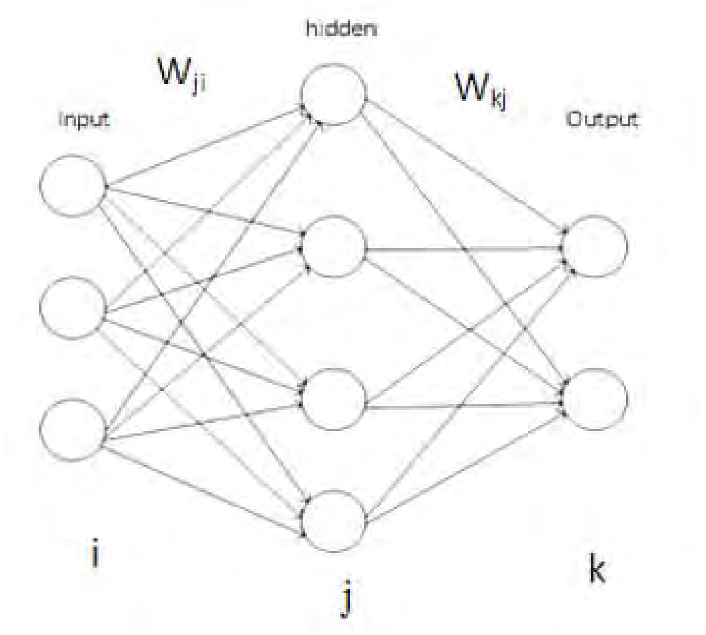

Fig 3. Example of the architecture of a MLP network

- Calculating global error:

The interest of this step is to calculate the global error observed (3) to the output network, which will be minimized to find the best possible correction for each synaptic weight.

$$E(n) = \frac{1}{2}\sum_{j=1}^{c}(d_j(n) - y_j(n))^2 \quad (3)$$

With:

$E(n)$ = the global error.

$d_j(n)$ = desired output.

$Y_j(n)$ = observed output.

C = output layer of the network.

To minimize this error, we must provide to each synaptic weight Wji the correction ΔWji according to the equation (4):

$$\Delta W_{ji} = -\eta \frac{\partial \mathrm{E}(n)}{\partial w_{ji}(n)} \quad (4)$$

The calculation of the equation (4) requires an activation function. Several activation functions exist in the literature, among them the Gaussian and the sigmoid functions (e.g. Fig. 4).

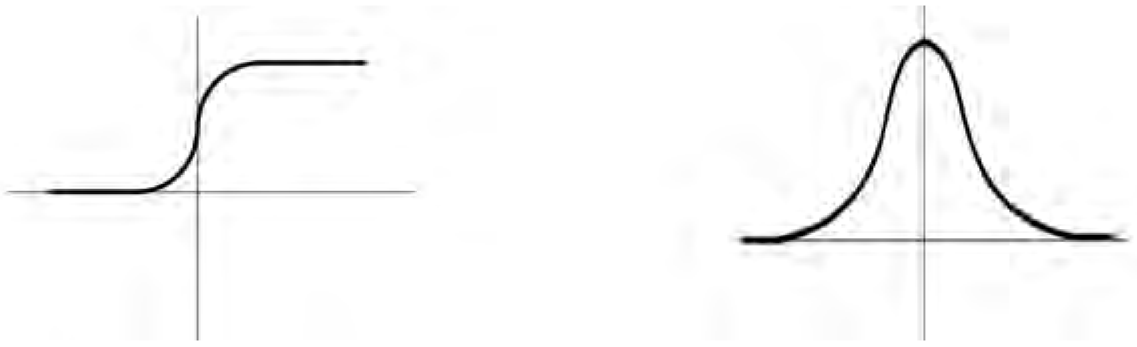

Fig 4. Sigmoid and Gaussian function

The transfer function most used in multilayer perceptron is the sigmoid function because it is an infinitely differentiable function, and the equation (5) shows the sigmoidal activation function and (6) its derivative:

$$f(x) = \frac{1}{1+e^{-x}} \quad (5)$$

$$f'(x) = f(x)(1-f(x)) \quad (6)$$

The difficulty in applying this rule is in the calculation of the partial derivative $\frac{\partial \mathrm{E}(n)}{\partial W_{ji}}$ which is not evident that only for the neurons of the output layer. To alleviate this difficulty, the calculation of this derivative is performed in the output layer to the input layer, where the name back- propagation comes. The synaptic weights of each neuron in the output and hidden layer are discounted using the delta rule: For the output layer (7) and for the hidden layer (8):

$$W_{ji}(n) = W_{ji}(n-1) + \eta \delta_j(n) y_i(n) \quad (7)$$

With: i = the last hidden layer.

δ_j = local gradient $(= e_j(n) y_j(n)(1-y_j(n)))$

$$W_{ji}(n) = W_{ji}(n-1) + \eta \delta_j(n) y_i(n) \quad (8)$$

With: i = previous layer.

$\delta_j(n) = y_j(n)(1-y_j(n)) \sum k$ hidden layer $\delta k(n) w_{kj}(n)$

In summary, the application of the algorithm on a multilayer perceptron requires two essential steps for calculating each element of learning base (e.g. Fig. 5):

- The first is the propagation of the input data of the first layer to the last (propagation).
- The second is the propagation of error signals of the last layer to the first (back-propagation).

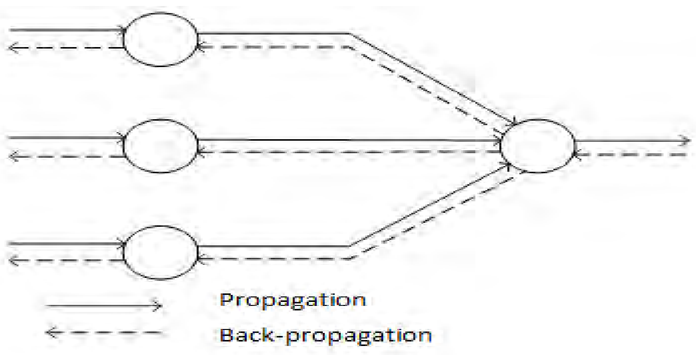

Fig 5. Propagation and back-propagation path

III.DETAILED DESIGN AND IMPLEMENTATION

This section describes in detail the various modules required for parallel and mixed implementation (serial and parallel) of MLP.

The multilayer perceptron in the learning and generalization phase manipulates data in the form of real numbers, which requires to be encoded according to a synthesizable format by the FPGA platform. There exist two possible representations: the fixed-point representation (FXP) and floating point (FLP).

- The FLP representation:

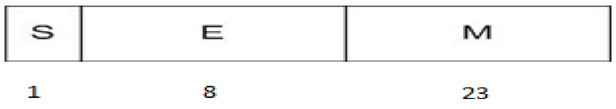

Fig 6. FLP representation

Generally, a FLP number is represented as: $X = (-1)^S \times 2^{E-127} \times 1,M$ with $0 < E < 255$, which is a representation of coding of real numbers (norm IEEE-754). This coding provides a standardized representation of real numbers, either in 32 bits (e.g. Fig. 6 [1]) (single precision), 64 bits (double precision) or 80 bits (extended).

- The FXP representation:

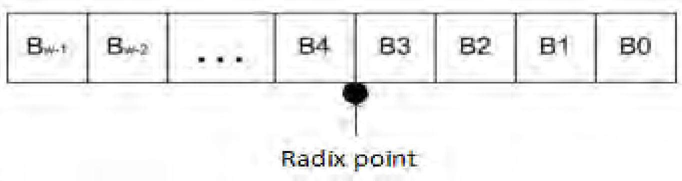

Fig 7. FXP representation

The FXP format is illustrated in (e.g. Fig. 7 [1]); it contains 2 parts, the integer part B_{w-1} to B_4 and the fractional part B_3 to B_0. The point is located at a definite site. It is a representation less used because it has a low precision because of lost digits to the right of the decimal point [1].

In this article, the data representation of the MLP in the form of real numbers concentrated on floating point representation at 32 bits, for an efficient MLP network in terms of accuracy.

A. *Implementation of a formal neuron*

The basic element of design of the MLP network with the VHDL hardware description language is the design of the neuron for transforming its inputs to the output according to the following steps:

1) *Multiplication of inputs by its synaptic weight.*

2) *Summation of multiplication result.*

3) *Transfer the summation with a sigmoid activation function.*

The design of formal neuron with the VHDL hardware description language can perform very complicated and difficult mathematical functions to implement in FPGA platform, as division and exponential which are used in the sigmoid function. The solution proposed in this paper allows implementing blocks megafunctions that have given the perfect solution while designing. The megafunctions are specific blocks that use effective methods of designs to create complex applications. These are blocks offered by the constructors of software simulation QUARTUS which is ALTERA software for the analysis and synthesis of the HDL designs. It also allows compiling designs and simulating design results in terms of resources used and execution time. These blocks named IP (Intellectual Properties) enable the synthesis complex functions as (memory, multiplier, comparator, etc. ...).

The megafunctions include complex functions useful for the design of formal neuron and artificial neuron network; they offer an appreciable gain in design time compared to coding a new block and provide access to the functionality of the internal architecture of megafuctions; registers and other simple functions, and the choice of representation of data in 32 bits or 64 bits.

The design of the formal neuron requires the use of several megafunctions blocks, among these blocks [12] [13]:

- Addition: block megafunction that implements the functions of addition and subtraction with a floating point representation of the IEEE-754 standard.
- Multiplication: block megafunction that implements the functions of multiplication.
- Exponential: it is an interesting megafunction block when designing because it implements the functionality of the complex exponential function.
- Division: block megafunctions also interesting because it implements the functions of division and inversion.

The combination of these blocks has helped design the formal neuron with sigmoid activation function according to the following scheme (e.g. Fig. 8 [12][13]):

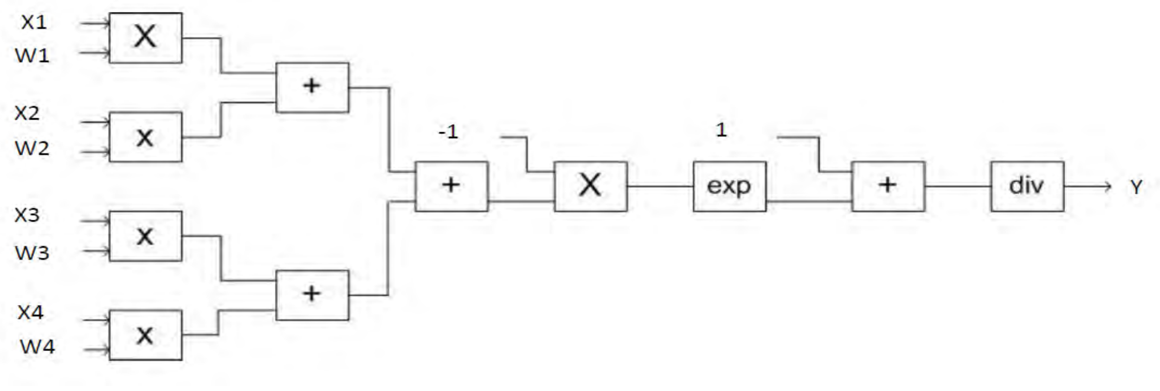

Fig 8.design of the formal neuron

In the rest of article, the formal neuron used in the design of artificial neuron network will be modular as shown in the following scheme (e.g. Fig. 9):

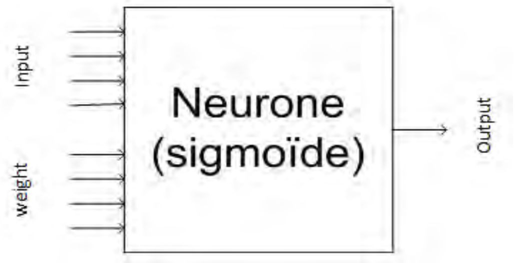

Fig 9. Schema of the modular formal neuron

B. Parallel implementation

The first implementation of artificial neuron network is a parallel implementation; artificial neuron network is a network that interconnects several neurons to achieve a well-defined task. This implementation requires a formal neuron with sigmoid function, which manipulates data in 32 bit floating point FLP, and then connects these neurons according to desired architecture. Each ANN architecture requires an input layer, one or more hidden layers (a single hidden layer is recommended [11]), and an output layer. In the implementation of ANN, the outputs of the neurons of a layer represent the inputs of the neurons of the next layer.

Example (e.g. Fig. 10): Implementing an architecture containing 3 neurons in the hidden layer and 2 in the output layer.

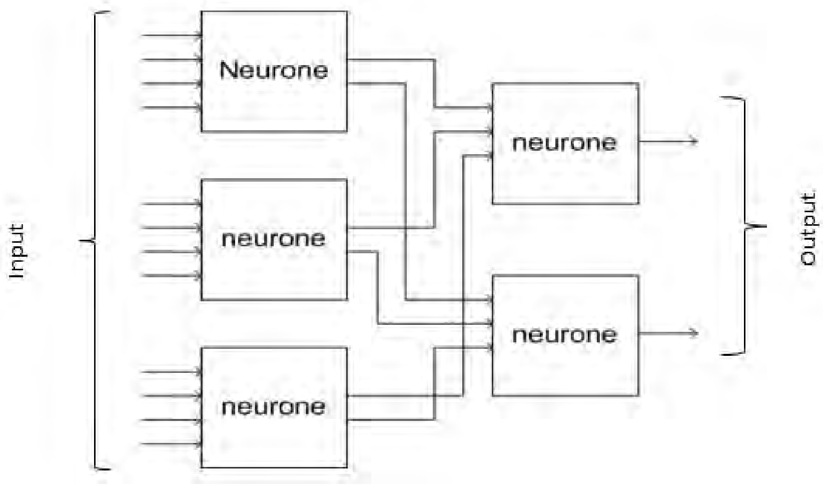

Fig 10. Design of the parallel implementation of the MLP

The specific constraints of this method in FPGA platform is the consummation of the memory area, we must take into consideration that the use of megafunctions blocks consumes a lot of space in the FPGA platform.

C. Mixed implementation

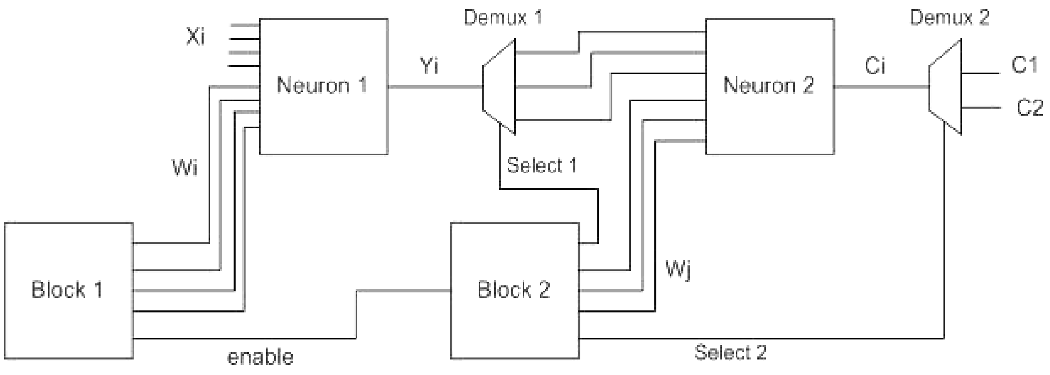

Fig 11.design of the mixed implementation of the MLP

The second implementation is a mixed implementation (serial and parallel), which all the layers (hidden and output) contain a single neuron which deals with the complete calculation of the layer, to build an artificial neurons network capable of performing a specified task. The interconnection of these neurons (layers) requires a number of blocks to guarantee both the timing and the proper functioning of the MLP (e.g. Fig. 11).

1) Block 1

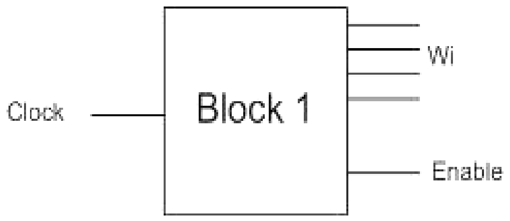

Fig 12. Schema of the Block 1

This block has two functions: the first is to generate the synaptic weights memorized in the memory of all the neurons in the hidden layer (W_i) at times well defined and the second is to choose the right moment to activate the block 2. This requires working with a block that is both a counter and a memory. As soon as the neuron, which represents the hidden layer, Receives the vector X_i at its input from the input layer, this block, generates the memorized synaptic weights corresponding to the first neuron, then successively generates the synaptic weights of the other neurons of this layer based on clock edges. With regard to the activation signal of the second block, block 1 counts the number of clock cycle required for calculating the output Y_i to enable the block 2.

2) Block 2

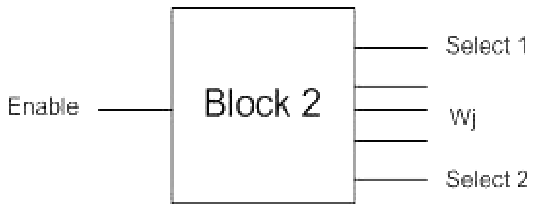

Fig 13.schema of the Block 2

The functioning of this block is similar to block 1. As soon as this block receives the activation signal from the block 1, meaning that the outputs of the hidden layer are available, the block 2 controls demux 1 which will look after to routing the output Y_i arriving in series to the inputs of the second neuron representing the output layer. Block 2 then generates, in the same manner as the block 1, the synaptic weights memorized. When the calculation is completed for this neuron, this block controls demux 2 which can display output to C1 and C2.

IV. TESTS AND RESULTS

The two VHDL implementations of artificial neuron network are synthesized on a FPGA platform of the family Altera Cyclone II Version EP2C70F896C6. To test the design, we have performed several tests on this network; these tests provide to the network all the necessary values (Synaptic weights and the inputs of the network). The test has been performed on a simple example that provides the network a vector of 4 parameters and 2 outputs. The topology chosen for this network is (4, 3, 2), 4 inputs, 3 neurons in the hidden layer and two neurons in the output layer, and this is valid for two implementations (parallel and mixed). The two ANN architectures are capable of test one or more examples not seen in the learning phase of the ANN; this requires one or more test vector for the example tested and final synaptic weights as shown in Table 1. These synaptic weights are determined during the learning phase, which provides the best synaptic weights to simulate this application. This phase was performed in the same network in C + + on Linux and has allowed to obtain the synaptic weights that will be exploited in the VHDL network. The simulation of the synthesized networks is performed using the option VECTOR WAVEFORM of Quartus II 9.1 software.

TABLE I.
The example tested and final synaptic weights

		Value	**Floating point representation in 32 bits (hexadécimal)**
Neuron 1 of the hidden layer	Wi11	-0.893062	BF649FA4
	Wi12	2.894112	40393921
	Wi13	-1.056650	BF87404E
	Wi14	3.849348	3F765AEE
Neuron 2 of the hidden layer	Wi21	-3.279809	BF51DB22
	Wi22	6.124747	40C3F7CE
	Wi23	-3.293418	C052C083
	Wi24	9.450170	41173333
Neuron 3 of the hidden layer	Wi31	-3.157932	C04A0C49
	Wi32	2.302409	401353F7
	Wi33	-3.287087	C0522F1A
	Wi34	2.299887	401322D0
Neuron 1 of the output layer	Wj11	0.827811	3F53B6C8
	Wj12	6.815142	40DA154C
	Wj13	-5.542226	C0B159B3
Neuron 2 of the output layer	Wj21	-0.822810	3F52A305
	Wj22	-6.821629	C0DA4A8C
	Wj23	5.468048	40AEF9DB
Output 1 of the two networks	C1	0.846	3F589374
Output 2 of the two networks	C2	0.154	3E1DB22D

The synaptic weights in Table 1 are used in the two architectures of artificial neuron network, in the parallel architecture as input for each neuron, and in the second mixed architecture the synaptic weights are stored in the two blocks and in the second mixed architecture synaptic weights are stored in the two blocks, so that they are provided to the neurons which represent the hidden layer and the output layer. The table also shows 2 outputs C1 and C2 when testing two implementations.

TABLE II.
The result of comparison between the parallel and mixed implementation

		Parallel architecture	**Mixed Architecture**
Execution time		1.78 us	1.9 us
Surface in the FPGA platform	Registre	25989 (38%)	9887 (14%)
	Combunational functions	34492 (50%)	13889 (20%)
	Logic element	39202 (57%)	15438 (23%)

The result of two implementations allows comparing the two architectures in terms of hardware resources and execution time. The analysis of this comparison illustrated in Table 2 show the following two points:

In terms of resources as shown in Table 2, parallel architecture requires a large space of the platform relative to the mixed architecture. By cons, in terms of execution time, the parallel architecture is relatively faster than the mixed architecture. However, this time difference is negligible because we are talking about a few nanoseconds. The mixed architecture that implements a multilayer perceptron remains an ideal architecture since it offers advantages of implementations in terms of resources unlike to the parallel architecture whose development is limited.

V. CONCLUSION

This paper has examined the technical of hardware implementation of an artificial neuron network (MLP type) in the FPGA platform, and the impact of the use of the data represented in floating-point on 32-bit on precision of calculations. The document also gave comparative results between two implementations (parallel and mixed), which allowed to highlight the performance of the mixed hardware implementation of the MLP network.

REFERENCES

[1] Savich, A. W., Moussa, M., &Areibi, S. (2007). The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study. Neural Networks, IEEE Transactions on, 18(1), 240-252.

[2] Mellit, A., Mekki, H., Messai, A., &Kalogirou, S. A. (2011). FPGA-based implementation of intelligent predictor for global solar irradiation, Part I: Theory and simulation. Expert Systems with Applications, 38(3), 2668-2685.

[3] Y. Sun, A.C. Cheng. Machine learning on-a-chip: A high-performance low-power reusable neuron architecture for artificial neural networks in ECG classifications .Computers in Biology and Medicine 42 (2012) 751–757.

[4] Roy Chowdhury, S., &Saha, H. (2010). Development of a FPGA based fuzzy neural network system for early diagnosis of critical health condition of a patient. Computers in biology and medicine, 40(2), 190-200.

[5] G.J. Gibson, S. Siu, C.F.N. Cowan, Application of multilayer perceptrons as adaptive channel equalizers, in: Proc. IEEE Conf. Acoust. Speech Signal Process., 1989, pp. 1183–1186.

[6] Gomperts, A., Ukil, A., &Zurfluh, F. (2011). Development and implementation of parameterized FPGA-based general purpose neural networks for online applications. Industrial Informatics, IEEE Transactions on, 7(1), 78-89.

[7] Tisan, A., &Cirstea, M. (2013). SOM neural network design–A new Simulink library based approach targeting FPGA implementation. Mathematics and Computers in Simulation, 91, 134-149.

[8] Rostro-Gonzalez, H., Cessac, B., Girau, B., & Torres-Huitzil, C. (2011). The role of the asymptotic dynamics in the design of FPGA-based hardware implementations of gIF-type neural networks. Journal of Physiology-Paris,105(1), 91-97.

[9] Lin, C. J., & Tsai, H. M. (2008). FPGA implementation of a wavelet neural network with particle swarm optimization learning. Mathematical and Computer Modelling, 47(9), 982-996.

[10] Grossi, G., &Pedersini, F. (2008). FPGA implementation of a stochastic neural network for monotonic pseudo-Boolean optimization. Neural networks, 21(6), 872-879.

[11] Sibanda, W., & Pretorius, P. (2011). Novel Application of Multi-Layer Perceptrons (MLP) Neural Networks to Model HIV in South Africa using Seroprevalence Data from Antenatal Clinics. International Journal of Computer Applications, 35.

[12] Atibi, m., Bennis, a., & Boussaa, m. (2014) precise calculation unit based on a hardware implementation of a formal neuron in a fpga platform. International Journal of Advances in Engineering & Technology (IJAET), Volume 7 Issue 3, pp. 733-742, July 2014.

[13] Boussaa M, Bennis A, Atibi M. (2014) Comparison between Two Hardware Implementations of a Formal Neuron on FPGA Platform. International Journal of Innovative Technology and Exploring Engineering, Volume-4 Issue 1.