IC Design Implementation of an Artificial Neuron Using Open-Source Tools

Marcelo Felix Carlos
Department of Electrical Engineering, Federal Institute of Education, Science and Technology, São Paulo (SP), Brasil
marcelo.carlos1997@gmail.com

Sara Dereste dos Santos
Department of Electrical Engineering, Federal Institute of Education, Science and Technology, São Paulo (SP), Brasil
sarad@ifsp.edu.br

Ricardo Pires
Department of Electrical Engineering, Federal Institute of Education, Science and Technology, São Paulo (SP), Brasil
ricardo_pires@ifsp.edu.br

***Abstract*—Nowadays, studies have shown that the hardware infrastructure as we know it has not been sufficient for the advancement of machine learning algorithms. To overcome this deficiency, the use of ASICs has been considered as the best alternative. However, due to the high development cost of both design and manufacture, the use of open-source software has proved to be an alternative to reduce costs. In this sense, the present work aims to demonstrate conceptually the implementation of a neurochip, using the MBP software for the Machine Learning algorithm and the Qflow package for the VLSI implementation, all of them open-source tools.**

Keywords—TCAD, ASIC, Open-Source, Machine Learning, Neurochip

I. Introduction

Although the use of ASICs (Application Specific Integrated Circuits) presents several advantages, not only in the field of Artificial Intelligence (AI), but also in several other applications [1] [2] [3], some limitations make the use of this kind of hardware the last solution to be sought. The main factor for this is the high cost associated to the design and production of an ASIC [4].

As a way to reduce costs in VLSI (Very Large Scale Integration) projects and research, the open-source movement proved to be the best alternative for universities and small companies to keep themselves in the business. Friesenhahn [5] proposes a VLSI digital design methodology, in addition to a set of free EDA (Electronic Design Automation) tools that the University of Texas Applied Research Laboratories (The University of Texas at Austin - ARL: UT) has evaluated and is now adopting. Stine [6], on the other hand, proposes an open-source PDK (Process Design Kit) to be used in universities and to facilitate teaching VLSI to students. Thapa [7] describes the characterization of the standard cell library developed by the University of Oklahoma (Oklahoma State University).

Therefore, the following work aims to present the development of a dedicated integrated circuit for artificial intelligence, using only open-source software, as an alternative to reduce costs in the development of ASICs.

II. Methodology

Considering the primary objective of this work, that is, the VLSI implementation of an AI algorithm, it was decided to simplify the development of the stage linked to machine learning. With that in mind, the following requirements were defined:

- For training the neural network, a dataset already known and of low complexity must be used;
- The neural network structure to be used would be the Rosenblatt Perceptron (artificial neuron) [8];
- The learning algorithm (known as back-propagation) would not be implemented in the ASIC, but only the network already trained, requiring a separate program for training the network and an interface for transferring the weights.

Regarding the requirements from a hardware point of view, the following aspects were defined:

- The VLSI implementation should be automatic, that is, starting from a hardware description language program, a software should synthesize the code at the gate level;
- Given the low complexity of the network to be implemented and in order to reduce the chip area, the data used should be represented with a maximum size of 8 bits;
- At the end of the VLSI synthesis process, a Graphic Database System (GDS2) type file should be generated. In addition, there should be the possibility of the final structure obtained to be simulated in order to prove its functionality.

A. Machine Learning algorithm implementation

The dataset chosen for the development of machine learning algorithm was the Iris database [9]. Provided by the University of California at Irvine (UCI), it is a database consisting of four characteristics: petal size, petal width, sepal size, and sepal width; of three flower species (Iris Setosa, Iris Versicolor and Iris Virginica) having 150 data instances or 50 instances per species.

The software chosen for training the developed networks was the MBP – Multiple Back Propagation [10]. This choice was made taking into account two factors: the software being open-source and the possibility of training the Perceptron-type network.

Before starting the training process, the database was divided as follows: two sets, being a set for training and a set for testing the network. The test set was formed by five examples of each species from the main database, while the training set with the rest of the examples. Therefore, 90% of the data were used for training and 10% for testing.

Considering the structure of the Rosenblatt Perceptron, one must pay attention to the limitation of this type of Perceptron in classifying only linearly separable patterns. Due to this fact, the training and testing sets were modified so that the network was trained to identify only one of the species in the presence of others. Therefore, the value "1" was assigned to all examples of a single species, while the others were assigned the value "0". The choice of these values is given by the activation function used in Perceptron, the sigmoid function (1):

$$\emptyset(x) = \frac{1}{1+e^{-x}} \tag{1}$$

The choice of the sigmoid as an activation function is due to its simplicity during the normalization of input data. Although this normalization is not a necessary process for training the network, this pre-processing speeds up the training [8] and is intrinsically present in the software used for training the neural networks. Equation (2) shows the normalization used by the MBP – "*min-max*" normalization.

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A \tag{2}$$

where v is the value to be normalized, min_A and max_A are the minimum and maximum values of the set of v. For the normalization to have effects on learning acceleration, it is necessary that the new minimum and maximum value – min_A and max_A – to be within of the range of the activation function used, represented by new_max_A and new_min_A. The "*min-max*" normalization for the sigmoid is shown in (3):

$$v' = \frac{v - min_A}{max_A - min_A} \tag{3}$$

All the training process is done at the MBP, being up to the user select the datasets used on the training and the test stage. After that, the weights are automatically generated and can be exported on a CSV (Comma Separated Values) format. For the training, a learning rate of 0.7 was used and an activation threshold value of 0.5 was considered.

B. *VLSI implementation*

Considering the basic structure of a Perceptron, the main elements that compose it are the weights, the summing element and the activation function as shown in Fig. 1.

For the implementation of the Perceptron, in addition to its basic structure, it would also be necessary to normalize the input data, due to the use of the MBP and the addition of a clock input to make the module synchronous and, thus, to avoid the occurrence of output glitches. Considering those factors, Fig. 2 shows the block diagram planned for the implementation.

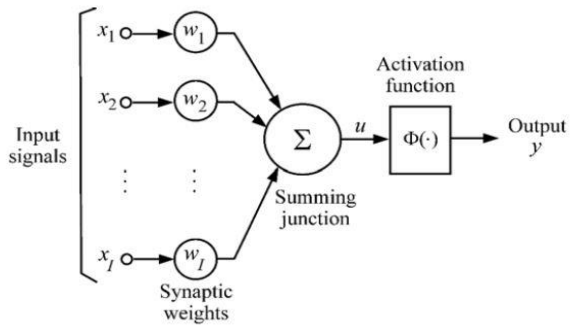

Fig. 1: Perceptron's structure

In view of the reduced number of bits to represent the input data, a fixed-point representation approach was chosen, since for the floating-point representation, according to the IEEE-754 standard, at least 32 bits would be required. Furthermore, as Adam [11] demonstrates, comparing the fixed-point representation with the single precision and double precision floating-point formats, it turns out that in fixed-point there is lower latency, power consumption and smaller chip area, despite the error rate is greater for this representation.

Also according to Adam [11], there are numerous forms of data representation by fixed point. For this work, the scaling of values by a factor of 10 was chosen, that is, the suggested system would have the capacity to represent one digit after the decimal point.

Despite the advantages presented in the use of fixed-point representation, there is a need for adaptations after multiplication operations, which require changes in the treatment of data in later stages. This happens because in the multiplications the scaling is doubled, that is, for the 10 factor scaling used, after the multiplication with t and n operands, we would have a factor of 100, as shown in (4):

$$(t \times 10) \times (n \times 10) = 100 \times (t \times n) \tag{4}$$

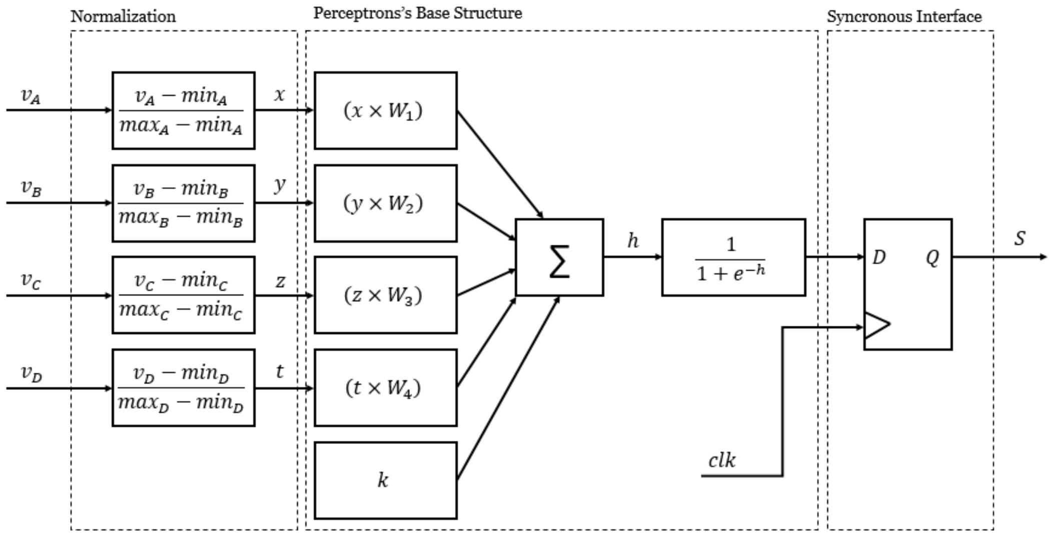

Fig. 2: Planned system block diagram

Furthermore, both the activation function and the normalization steps are not directly synthesizable in hardware description code, requiring dedicated structures to perform these operations. To get around this situation, the following strategies were used:

- For the activation function, it was chosen to assemble a LUT (lookup table) to represent the sigmoid function, as is done in [12], [13] and [14];
- To circumvent the division operations of the normalization step, such process was adapted together with the multiplication by a weights block, in order to obtain a parameter as a function of the weight and denominator of the normalization function, called **α**.

Equation (5) shows the process for obtaining the **α** parameter:

$$\alpha = \frac{W}{(max_A - min_A)} \tag{5}$$

For the system in Fig. 3, its operation is described as follows: after training the network, the weights (*W*) are transformed into the **α** parameter using (5) together with the minimum (*min$_A$*) and maximum (*max$_A$*) values in the database for each characteristic. Those values are then scaled, and passed to the system. Finally, test data entries are also scaled and passed to the system. When a clock pulse occurs, the system displays the rating value on the output. After having the system described in Verilog, its layout synthesis process begins.

For the synthesis, the Qflow package was used, which integrates all the necessary tools in a digital synthesis design flow. Its design flow is divided in 11 steps, allowing the designer to check the product and logs of each step and change configuration parameters, restarting the step if judges necessary, or even applying manual changes. The following open-source tools are part of the Qflow package [15]:

- Verilog parser, high-level synthesis, and logic optimization and verification (yosys);
- Placement (graywolf);
- Detail Router (qrouter 1.4);
- Static Timing Analysis (vesta);
- Layout viewer and editor (Magic 8.1).

For the synthesis, the standard cell library based on the MOSIS scalable CMOS rules for the TSMC process with 0.18**µ**m technology from the Oklahoma State University was used [16].

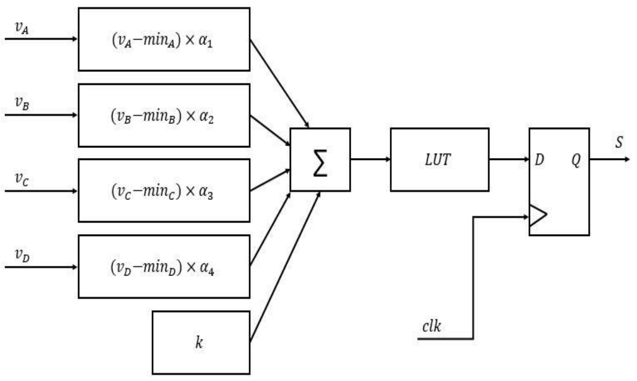
v_A
$(v_A - min_A) \times \alpha_1$
v_B
$(v_B - min_B) \times \alpha_2$
v_C
$(v_C - min_C) \times \alpha_3$
v_D
$(v_D - min_D) \times \alpha_4$
k
$\sum$
LUT
D
Q
S
clk

Fig. 3: Implemented system block diagram

III. Results

As a result of the training stage, a classification error of 0.099% was obtained for the training set and an error of 0.077% for the test set (considering the square root of the mean square error). Fig. 4 shows the results of the classification of the test set. The red trace shows the classifications index obtained against the black trace indicating the expected result. The axis line shows the samples of the test set, being the first five of *Iris-Setosa* and the remaining samples from the other two species. For a succeeded classification, *Iris-Setosa* samples must reach an index as close as possible to “1” while the others to “0”.

After the conversions of the weights obtained in the training of the algorithm in the MBP, necessary due to the

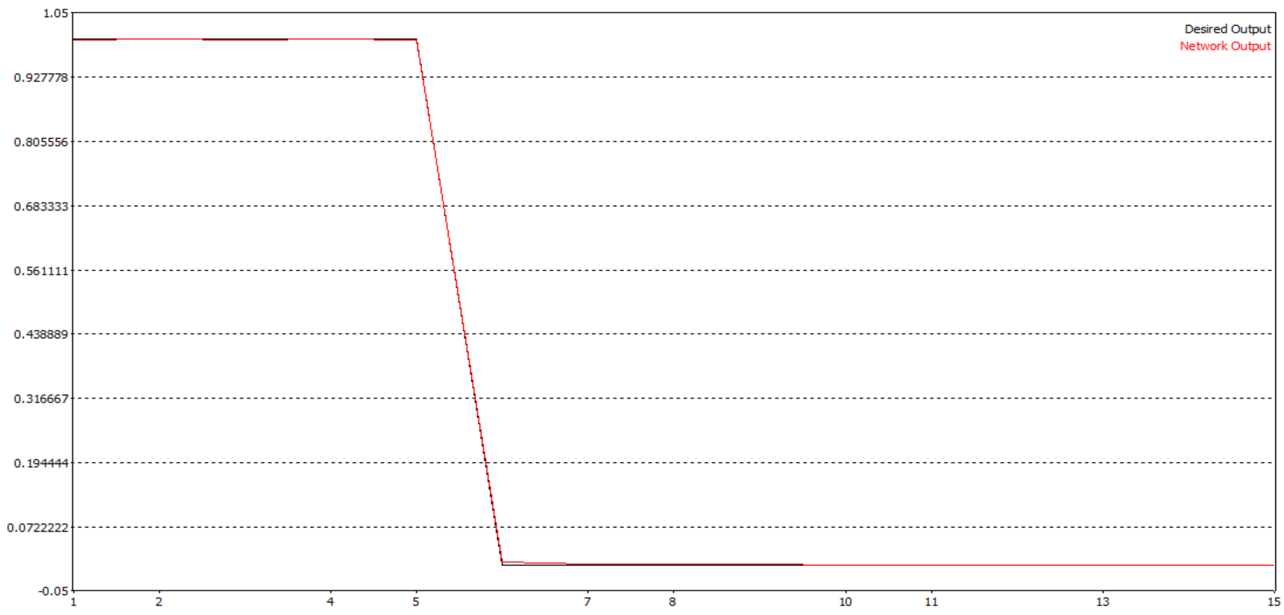

Fig. 4: Classification results obtained for the test dataset – MBP graphic view

restrictions of data representation, scaling and adaptation of the normalization in the system, these are applied to the system described in Verilog and simulated. Calculating the square root of the root mean square error for the results obtained in the simulation, it is found that, after the transcription of the system in Verilog, it reached an error of 2.6%. Fig. 5 shows the results obtained for the Verilog simulation of the system. As for the results obtained previously, the first five samples are from the *Iris-Setosa* species, while the remaining samples from the other two species. At each clock pulse, the values of petal size (r_in1), petal width (r_in2), sepal size (r_in3), and sepal width (r_in4) are passed to the system producing an output (w_output) corresponding to the index obtained in the classification. As a product of the use of data scaling, both inputs and output work with a multiplicative factor of 10, meaning now that *Iris-Setosa* samples must reach an index as close as possible to "10" while the others to "0".

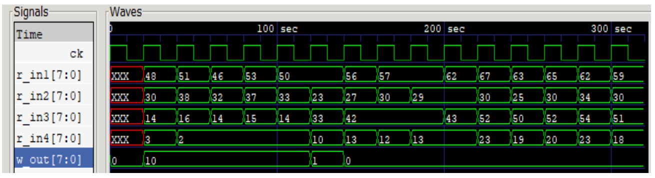
Signals
Waves
Time
0
100 sec
200 sec
300 sec
ck
r_in1[7:0] XXX 48 51 46 53 50 56 57 62 67 63 65 62 59
r_in2[7:0] XXX 30 38 32 37 33 23 27 30 29 30 25 30 34 30
r_in3[7:0] XXX 14 16 14 15 14 33 42 43 52 50 52 54 51
r_in4[7:0] XXX 3 2 10 13 12 13 23 19 20 23 18
w_out[7:0] 0 10 1 0

Fig. 5: System simulation results obtained for the description in Verilog

The same error is obtained after the synthesis of the code from behavioral to structural, indicating that the synthesis produced faithful results.

From the logs generated during the layout synthesis process, it was verified that there were no errors in the DRC and LVS verification processes. Furthermore, analyzing the results of the STA simulation through the logs generated in this step, it is estimated that the maximum operating clock time of the designed chip is 170MHz and the final area occupied by the layout is 0.15785302 mm². Fig. 6 shows the layout design obtained by Magic VLSI editor, which is part of the Qflow package.

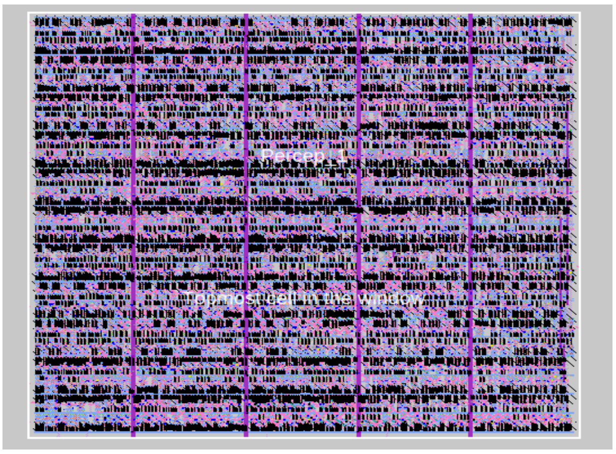

Fig. 6: Designed chip Layout (GDS2) –Magic VLSI view

IV. Conclusion

In this work, it was shown that it is possible to develop an ASIC for a machine learning application using only open-source software. It is considered that the results of accuracy, of 97.4%, and final area of the chip, of 0.15785302 mm², are satisfactory, taking into account the objective of validating the tools used.

As a future work, it is intended to test the same architecture presented in this work against market tools such as the Synopsys compiler, for comparison in relation to the results achieved by the Qflow package. In addition to validating the tools, this work also allows for the improvement of the presented architecture, from the elements of the Perceptron structure, as well as verification of other data representations. Not limited to the architecture, it can also provide a basis for more complex neural network layouts to be designed.

References

[1] SZE, Vivienne. Designing hardware for machine learning: The important role played by circuit designers. IEEE Solid-State Circuits Magazine, v. 9, n. 4, p. 46-54, 2017.

[2] NURVITADHI, Eriko et al. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2016. p. 1-4.

[3] NURVITADHI, Eriko et al. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In: 2016 International Conference on Field-Programmable Technology (FPT). IEEE, 2016. p. 77-84.

[4] KOTTOLLI, Arun. The economics of structured-and standard-cell-ASIC designs. EDN Magazine, v. 51, n. 6, p. 61-68, 2006.

[5] FRIESENHAHN, Russell; YORK, Johnathan. ARL: UT's Experiences in the Free Open-Source VLSI EDA Landscape. 2018.

[6] STINE, James E. et al. FreePDK: An open-source variation-aware design kit. In: 2007 IEEE international conference on Microelectronic Systems Education (MSE'07). IEEE, 2007. p. 173-174.

[7] THAPA, Rabin; ATAEI, Samira; STINE, James E. WIP. Open-source standard cell characterization process flow on 45 nm (FreePDK45), 0.18 µm, 0.25 µm, 0.35 µm and 0.5 µm. In: 2017 IEEE International Conference on Microelectronic Systems Education (MSE). IEEE, 2017. p. 5-6.

[8] HAYKIN, Simon. Neural Networks and Learning Machines. 3/E. Pearson Education India, 2010.

[9] DUA, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[10] LOPES, Noel. Multiple Back-Propagation (with CUDA). [Online] Available: https://sourceforge.net/projects/mbp/files/latest/download. [Accessed Oct. 29, 2019].

[11] ADAM, Norbert et al. The Impact of Data Representations on Hardware Based MLP Network Implementation. Acta Polytechnica Hungarica, v. 15, n. 2, 2018.

[12] MEHER, Pramod Kumar. An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks. In: 2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip. IEEE, 2010. p. 91-95.

[13] LEBOEUF, Karl et al. High speed VLSI implementation of the hyperbolic tangent sigmoid function. In: 2008 Third International Conference on Convergence and Hybrid Information Technology. IEEE, 2008. p. 1070-1073.

[14] NAMIN, Ashkan Hosseinzadeh et al. Efficient hardware implementation of the hyperbolic tangent sigmoid function. In: 2009 IEEE International Symposium on Circuits and Systems. IEEE, 2009. p. 2117-2120.

[15] Open Circuit Design. Qflow. [Online] Available: http://opencircuitdesign.com/qflow/. [Accessed Oct. 18, 2019].

[16] Oklahoma State University. VLSI Computer Architecture Research Group. [Online] Available: http://opencircuitdesign.com/qflow/. [Accessed Oct. 18, 2019].