IMPLEMENTATION OF A NEURON MODEL USING FPGAS

M. A. Bañuelos-Saucedo, J. Castillo-Hernández, S. Quintana-Thierry, R. Damián-Zamacona, J. Valeriano-Assem, R. E. Cervantes, R. Fuentes-González, G. Calva-Olmos & J. L. Pérez-Silva.

Laboratorio de Electrónica, Centro de Ciencias Aplicadas y Desarrollo Tecnológico, UNAM, Apdo. Postal 70-186, México, D.F.

Received October 10th 2001 and accepted January 7th 2003

ABSTRACT

Artificial neural networks base their processing capabilities in a parallel architecture, and this makes them useful to solve pattern recognition, system identification, and control problems. In this paper, we present a FPGA (Field Programmable Gate Array) based digital implementation of a McCulloch-Pitts type of neuron model with three types of non-linear activation function: step, ramp-saturation, and sigmoid. We present the VHDL language code used to implement the neurons as well as to present simulation results obtained with Xilinx Foundation 3.0 software. The results are analyzed in terms of speed and percentage of chip usage.

RESUMEN

Las redes neuronales artificiales basan sus capacidades de procesamiento en una arquitectura paralela, lo cual las hace útiles para resolver problemas de reconocimiento de patrones, identificación de sistemas y control. En este artículo, presentamos una implementación digital basada en arreglos de compuertas programables (FPGA, por sus siglas en inglés) de un modelo de neurona tipo McCullogh-Pitts con tres tipos de funciones de activación no-lineal: escalón, rampa-saturación y sigmoide. Presentamos el código en lenguaje VHDL utilizado para implementar las neuronas, y también presentamos los resultados de su simulación obtenidos con el software Xilinx Foundation 3.0. Los resultados son analizados en función de la velocidad y el porcentaje de utilización del chip.

KEYWORDS: Digital artificial neuron, field programmable gate array, McCullogh-Pitts neuron.

1. INTRODUCTION

Artificial neural networks (ANN) have been used successfully in pattern recognition problems, function approximation, control, etc. Their processing capabilities are based on a parallel architecture. There are different kinds of electronic implementations of ANN: digital, analog, hybrid [1]; and each one has specific advantages and disadvantages depending on the type and configuration of the network, training method and application.

For digital implementations of ANN there are different alternatives: custom design, digital signal processors, programmable logic, etc. Among them, programmable logic offers low cost, powerful software development tools and true parallel implementations. Field programmable gate arrays (FPGA) are a family of programmable logic devices based in an array of configurable logic blocks, which gives a great flexibility in the development of digital systems. For example, Virtex II series of Xilinx FPGAs have up to eight million gates. This amount of available resources for digital system design gives a great flexibility for implementing complex systems, such as a specific purpose microprocessor, network controllers, digital video systems, and for implementing digital ANNs.

In this paper, we present the design of an artificial neuron model based on a XC4000 Xilinx series of FPGAs [2]. The model comprises an input aggregation stage and an output activation function stage. The input stage is an 8-bit binary adder. Three types of activation functions were used: step, ramp-saturation, and sigmoidal. For implementing the

sigmoidal activation function, the approximation made by Kwan [3] was used. The results obtained will be used as a starting point for the generation of complex ANN for applications requiring of parallel computing.

For the implementation of the neuron model, VHDL language was used [4-6]. VHDL (**V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage) is a hardware description language which simplifies the development of complex systems because it is possible to model and simulate a digital system from a high level of abstraction and with important facilities for modular design.

2. NEURON MODEL

The used neuron model corresponds to the next generalization

$$y(t) = \psi\left[\mathbf{w}_a \copyright \mathbf{x}(t)\right] \tag{1}$$

where y(t) is the output of the neuron, $\mathbf{W}_a$ is the input weight vector, **x**(y) is the input signal vector and © is the confluence operator. $\psi[\bullet]$ is a non-linear activation function. A block diagram of the neuron model is shown in figure 1.

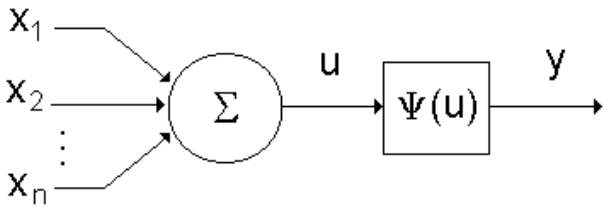

Figure 1. Block diagram of the neuron model.

3. STEP ACTIVATION FUNCTION

The step function is the simplest case of a non-linear activation function and is based on the "all-nothing" response of biological neurons. Step function can be defined as follows

$$y(u) = \begin{cases} 0 \text{ if } u < \theta \\ 1 \text{ if } u \geq \theta \end{cases} \tag{2}$$

where u is the sum of all input signals and θ is a threshold level.

In table I, it is shown the VHDL code used for the implementation of a neuron with step activation function.

Table I. VHDL code for a neuron with step activation function.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity escalon is
port(A,B: in integer range 0 to 15;
          UMBRAL: in integer range 0 to 31;
          SAL: out std_logic);
end entity escalon;

architecture behavioral of escalon is
signal TH : integer range 0 to 31;
signal A1, B1: integer range 0 to 31;
begin
A1<=A;
B1<=B;
TH<=UMBRAL;
SAL<='1'when A1+B1>TH else '0';
end architecture behavioral;
```

For simulating purposes two inputs (A and B) were defined as 4-bits, threshold as 5-bits, and output as 1-bit. Figure 2 shows the timing diagram for this neuron model. In this case, input A was fed with the signal of a binary counter while input B was fed with a constant value of '0010_{bin}'. Threshold level was set to '01000_{bin}'. When the sum of the inputs is greater than the threshold, the output becomes '1_{bin}'.

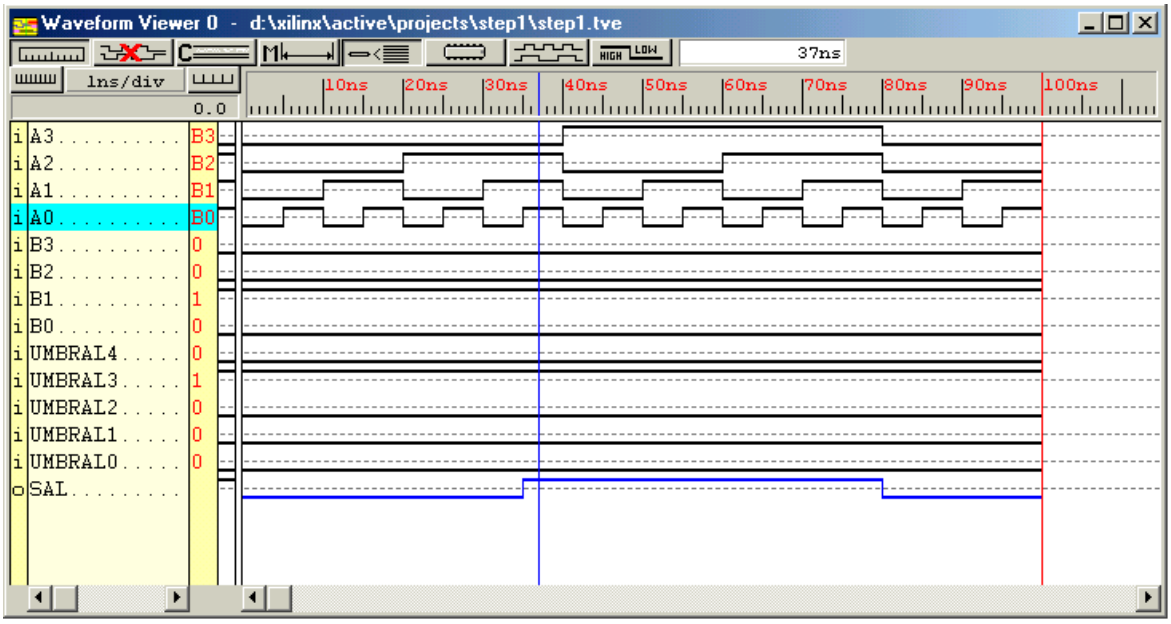

Figure 2. Timing diagram of a neuron with step activation function.

4. RAMP-SATURATION ACTIVATION FUNCTION

The ramp-saturation activation function produces an output gradually growing from zero to a maximum level. In order to minimize the hardware resources needed for this function, we implemented a unit slope ramp according to the next expression

$$y(u) = \begin{cases} u \text{ if } u < \theta \\ \theta \text{ if } u \geq \theta \end{cases} \quad (3)$$

where θ is the saturation level.

Figure 3 shows the plot of the ramp-saturation function. The VHDL code used to implement this neuron model is shown in Table II.

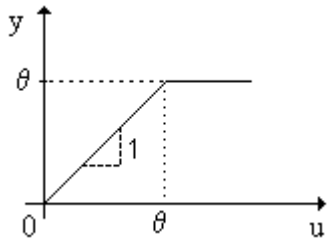

Figure 3. Ramp-saturation function.

Table II. VHDL code of a neuron with ramp-saturation activation function.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity ramp is
port(A,B: in integer range 0 to 15;
        SATURACION: in integer range 0 to
31;
        SAL: out integer range 0 to 31);
end entity ramp;

architecture behavioral of ramp is
signal TH,SUMA : integer range 0 to 31;
signal A1, B1: integer range 0 to 31;

begin

A1<=A;
B1<=B;
TH<=SATURACION;
SUMA<=A1+B1;

SAL<=SUMA when SUMA<TH else TH;

end architecture behavioral;
```

The resulting simulation timing diagram of the neuron with ramp-saturation activation function is shown in figure 4. Again, we have defined two 4-bits inputs (A and B) and a 5-bit saturation level input. Input A was fed with a binary counter and input B is held constant to '0010_{bin}'. The saturation level is set to '01000_{bin}'. As it can be seen, the output equals the sum of the inputs as long as it is lower to the saturation level. Once the sum equals or becomes greater than the saturation level, the output remains constant and equal to the saturation level.

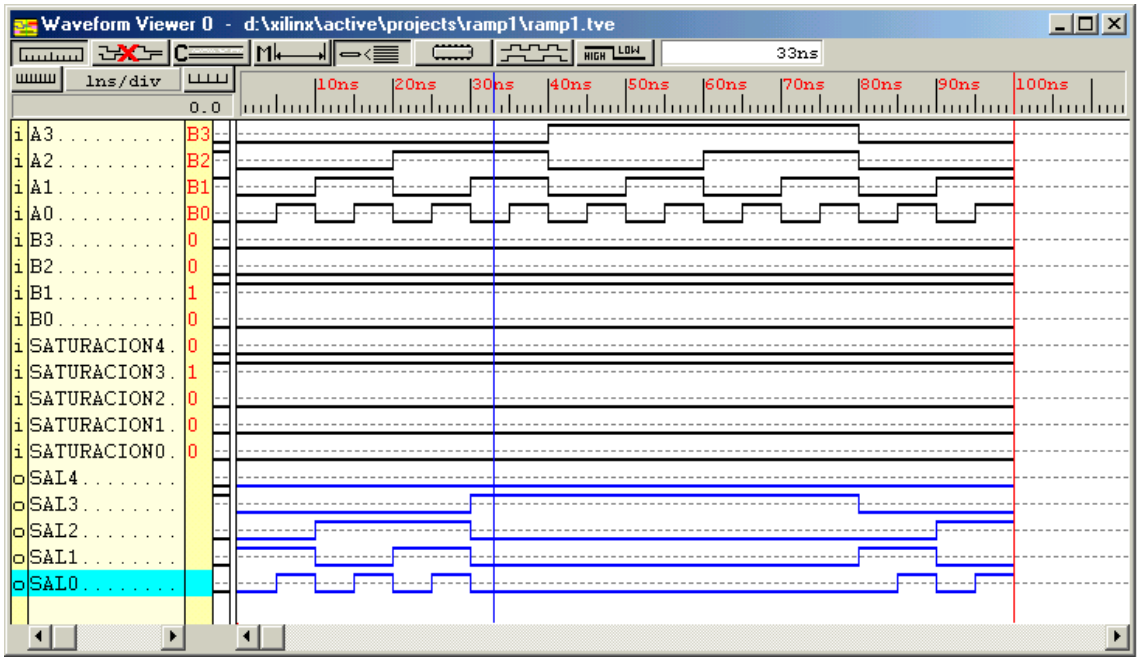

Figure 4. Timing diagram for a neuron with ramp-saturation activation function.

5. SIGMOIDAL ACTIVATION FUNCTION

Sigmoidal activation functions are widely used in ANN, due to their monotonic character and derivability which make them suitable for training algorithms. The sigmoid function is not easily implemented in digital hardware since it consists of an exponential series. Many digital implementations appeal to some sort of table lookup, but this approach still demands a great amount of hardware resources. As an alternative, we used the quadratic function proposed by Kwan [3], which is defined by

$$G_{S1}(z) = \begin{cases} 1 \text{ for } L \le z \\ H_{s1}(s) \text{ for } -L < z < L \\ -1 \text{ for} \le -L \end{cases} \quad (4)$$

where L depends on the level of saturation of the function, and $H_{s1}(s)$ is defined by

$$H_{s1}(z) = \begin{cases} z(\beta - \theta z) & \text{for } 0 \le z \le L \\ z(\beta + \theta z) & \text{for } -L \le z \le 0 \end{cases} \quad (5)$$

where β and θ are parameters for setting the slope and gain.

A comparison between bipolar sigmoid defined by (6) and Kwan approximation is presented in figure 5.

$$G_s(z) = (1 - e^{-\tau z})/(1 + e^{-\tau z}) \quad (6)$$

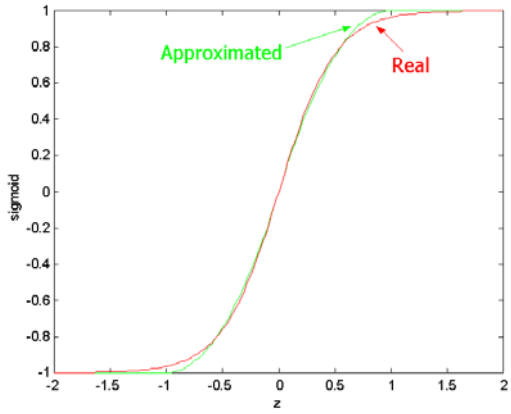

Figure 5. Comparison between bipolar sigmoid and Kwan approximation or an input ranging from 0 to 512_{10}

In table III, the VHDL code for an approximated sigmoidal activation function is shown. Parameters have been set for an input ranging from 0 to 512_{10}.

Table III. VHDL code of an approximated sigmoidal activation function.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Sigmoid1 is
   port (
      Z: in INTEGER range 0 to 511;
      SAL: out INTEGER range 0 to 255
   );
end Sigmoid1;

architecture Sigmoid1_arch of Sigmoid1 is
signal TEMP: integer range 0 to 255;
signal B1: integer range 0 to 511;
signal A1: integer range 0 to 511;
signal ZTHETA: integer range 0 to 65535;
signal ZZ : integer range 0 to 262144;
constant L : integer range 0 to 255:= 255;
constant M : integer range 0 to 1023:= 512;
begin
A1<=Z;
B1<=M-A1;
ZZ<=B1*A1;
ZTHETA<=ZZ/256;
TEMP<=ZTHETA;
SAL<=TEMP when A1<L else L;

end Sigmoid1_arch;
```

The results of the simulation for the sigmoidal activation function are shown in the timing diagram of figure 6. An 8-bit input (Z) was defined for the simulation and it was fed with a binary counter signal.

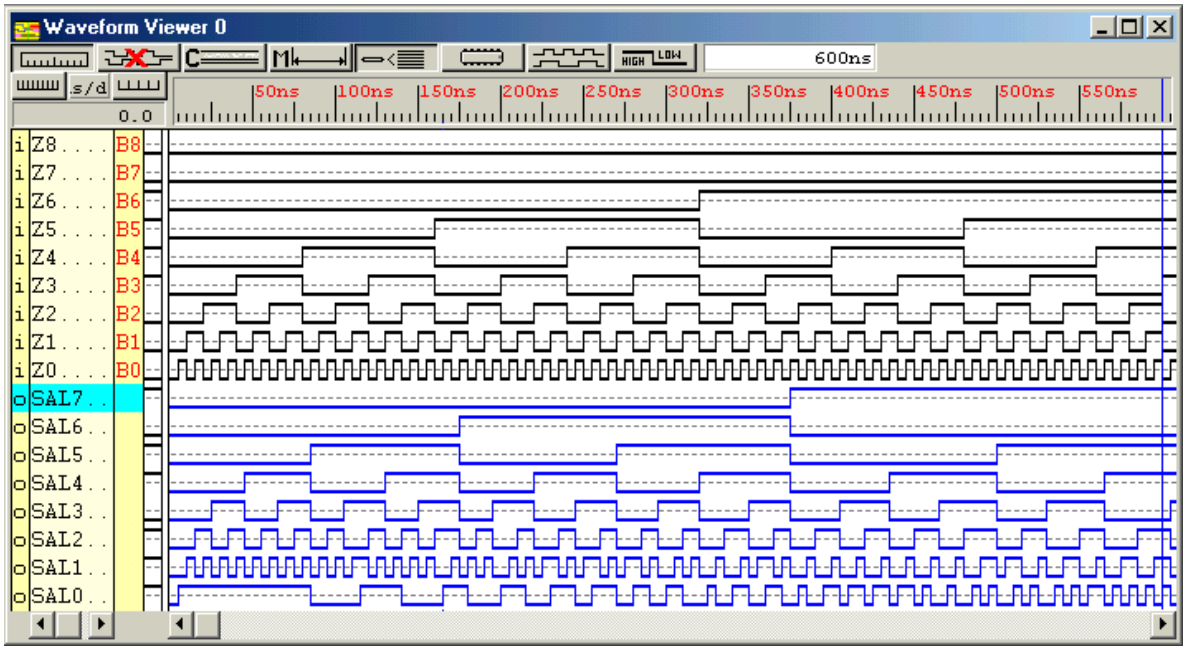

Figure 6. Timing diagram for the sigmoidal activation function simulation.

In order to make easier the interpretation of the results presented in figure 6, the output data file was edited and loaded in a spreadsheet. This allow us to generate the plot in figure 7, where it can be seen that the approximation is quite good.

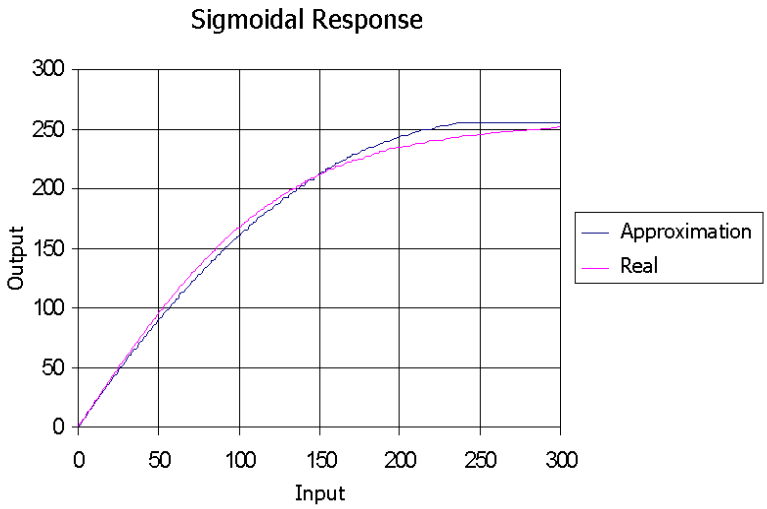

Figure 7. Comparison between a bilinear sigmoid and the Kwan approximation implemented in a FPGA.

6. RESULTS

We implemented the three models (step, ramp-saturation, and sigmoidal) in a XC4005ECP84-4 FPGA. The results obtained for the operation speed and the percentage of used resources are stated in table IV.

Table IV. Comparison of speed and resources used.

Function	Number of CLB used	Percentage of chip utilization	Maximum time delay between pins	Maximum working frequency
Step	5 de 196	2%	5.15 ns	194 MHz
Ramp	10 de 196	5%	7.62 ns	131 MHz
Sigmoidal	78 de 196	39%	16.65 ns	60 MHz

As it can be seen, the step function and the ramp-saturation function require a very few configurable logic blocks (CLB) in comparison with the sigmoidal function. The last one uses up to eight times the number of CLBs as the ramp-saturation which means that further optimization should be made in order to use it in multilayer ANN.

The operation speed in all cases is very good and it gives a clear idea of the advantages of using FPGAs, since multiple modules can be working in parallel with a minimum reduction in performance (due to the increased number of interconnections).

Since the more advanced families of FPGAs can contain more than 10,000 CLB, then it is clear that we can implement a network with an interesting number of neurons working in parallel in just a single chip. On the other hand, using high level languages for developing hardware, such as VHDL, represent a very practical option when dealing with complex systems. Finally, we can say that FPGAs constitute a very powerful option for implementing ANN since we can really exploit their parallel processing capabilities.

7. REFERENCES

[1] Bañuelos, M. A., *Diseño y construcción de una neurocomputadora analógica*. Tesis de Maestría. Facultad de Ingeniería, UNAM, 1997, 144 págs.

[2] Xilinx, *VHDL Reference guide.* Xilinx Inc., 1999.

[3] Kwan, H. K., "Simple sigmoid-like activation function suitable for digital hardware implementation". *Electronic Letters*. v.28, no. 15. July, 1992, pp. 1379 -1380.

[4] Yalamanchili S. *Introductory VHDL from simulation to synthesis.* Prentice May, 2001, 401 págs.

[5] Pardo-Carpio, Boluda-Grau F. & José A., VHDL Lenguaje para síntesis y modelado de circuitos. Alfaomega 2000, 238 pags.

[6] Xilinx *VHDL Reference guide.* Xilinx Inc., 1999.