FPGA implementation of Hebbian neural network for Engineering Educational

Marco A. Moreno-Armendariz[1], Osvaldo Espinosa Sosa[1] and Floriberto Ortiz Rodriguez[2]

[1] Centro de Investigación en Computación-IPN
CIC-IPN
AV. Juan de Dios Bátiz S/N, México D.F.,07738, México
marco_moreno@cic.ipn.mx
and
[2]Departamento de Control Automatico
CINVESTAV-IPN
A.P. 14-740, Av.IPN 2508, México D.F., 07360, México

Abstract. In this paper we are working in a new development of educational material for advanced courses in engineering. One of the hottest issues is the implementation of different computer intelligence algorithms in programmable logic technologies. We start this goal with the Hebbian neural network as an introduction of neural network courses. This article shows the design of neural networks using the Hardware Description Language VHDL and its implementation in Field Programmable Gate Arrays FPGAs. The code is totally open so that the user can verify the value of internal signals and make modifications to the structure of the design which can be very useful when a student is learning neural networks and digital design. The main advantages that are offered are that the design can be totally monitored and modified, is portable and it isn't necessary to view designs like a black box in where we do not know the internal structure of the system.

1 Introduction

Nowadays, programmable technologies have penetrated in many areas and places in where technology is present. One of the best cases is the Field Programmable Logic Array FPGA's , and the evolution of these devices has allowed to increase the density and capacity allowing integration of complete Systems on a Chip (SoC), in addition, these elements are offered in low cost and low power consumption devices. Designers have two options to obtain a design, the first one is the complete design of the circuit and second one is to purchase existing designs called CORE's that allows designers to use it as if they were a black box allowing to diminish dedicated effort. A disadvantage of the second option is that it implies the cost of acquisition of these CORE's. Although there are free ones, these usually are tied to a family of FPGAt's and this creates dependency. Another disadvantage is that they are not susceptible to be modified nor verified

Computer Engineering.
Research in Computing Science 30, 2007, pp. 81-88

its internal state. From an educational point of view, this is unacceptable for our requirements. On the other hand, design totally one circuit has the advantage to offer independence of companies, can be portable to any device of any company and with the characteristic to allow to students to use, monitor and even modify the characteristics of designs as much as it is required[1].

The designed circuit corresponds to a hebbian neural network. In the field of neural networks, the non supervised algorithm could be the denominated hebbian learning method that consists of increasing the value of weights that join two neurons if they activate simultaneously, and to diminish the value if they activate on a differential manner. A hebbian neural network can be arranged in a single layer or several: the inputs propagate to the internal layer, and when coming out, and after the propagation, the weights change in the indicated form. The hebbian learning is equivalent to an analysis of main components of inputs.

2 Hebbian neural network

In 1949, Donald Hebb gives the first learning law for an artificial neural network; the main idea is trying to represent how the brain learns at the cellular level [2]. In the brain, a neuron receives many inputs from a large number of other neurons through synaptic connections. Hebb's law states that if a neuron A is repeatedly activated by another neuron B, the neuron A will become more sensitive whenever both of them fire simultaneously. So, this learning can be viewed as strengthening a synapse according to the correlation between the activation levels of the neurons it connects. The neuron output signal $y_{net}(k)$ for the network in figure 1 can be expressed as [3],

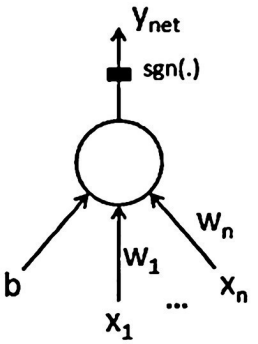

Fig. 1. Hebbian neural network model.

$$y_{net}(k) = sgn\left(\sum_{i=1}^{n} w_i(k)\, x_i(k) + b\right) \quad i = 1, \cdots, n$$

where $sgn(.)$ is the signum function. Given a set of training data $(x_i(k), y_i(k))$,the learning law used to update the weight values $w_i(k)$ and the bias $b(k)$ are,

$$w(k+1) = w(k) + x^T(k)y(k)$$
$$b(k+1) = b(k) + y(k)$$

Thus, if the correlation between the input and the output signals, which can be represented by the product $x^T(k)y(k)$ is positive, it enhances the strength of the synapse weight value. Otherwise, if it is negative, the weight value should be decreased [4].

This net is a linear associator network, since associate one or more pairs of vectors $(x(k), y(k))$ so that given $x(k)$ as the input, the network will produce $y_{net}(k)$. Furthermost, when a vector close to $x(k)$ is presented, the network will produce a vector close to $y_i(k)$. The representation is (consider $b(k) = 0$),

$$y_{net}(k) = x(k)w(k)$$

Suppose the network has learned to associate n pairs of vectors. That is,

$$w(k) = \sum_{i=1}^{n} x_i^T(k)y_i(k)$$

Futher, assume that the vectors $x_1(k), x_2(k), \cdots, x_n(k)$ are orthogonal and of unit length (i.e, they are orthonormal): If $j = i$, $x_j(k)x_i(k) = 1$;else it is 0. Then $x_j(k)$ can always be transformed into $y_j(k)$ by the network without error because

$$x_j(k)w(k) = \sum_{i=1}^{n} \left(x_j(k)x_i^T(k)\right) y_i(k) = y_j(k)$$

In a d-dimensional input space, the maximum possible number of orthogonal vector is d. Thus, the number of vector pairs that we can associate exactly is limited to d. If x_i's are not orthogonal, the we get an error when attempting to retrive $y_j(k)$ using $x_j(k)$. This error ϵ can be estimated by

$$\epsilon = \sum_{i \neq j} \left(x_j(k)x_i^T(k)\right) y_i(k)$$

In some cases, the weights $w(k)$ can be chosen so that the error ϵ is small.

3 Real time design

In order to make this work, it was used the software provided by Altera denominated Quartus II[5], this software allows to capture descriptions of circuits using the VHDL Hardware Description Language, after that, it makes a syntax revision of code. Once the description in VHDL is the correct one, a synthesis is made to transfer it to an equivalent design of gates and logic elements. It is

Input		Output
x_1	x_1	y
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Table 1. Training set for the learning phase.

important to mention that after the synthesis it is possible to use a simulator to verify the functionality of the circuit, that is, to verify that logically the design behaves in a correct form when the corresponding inputs are applied to it and the generated outputs corresponds to results expected by designers. When simulation is correct, it comes to make a process called "place and route" that has the objective to locate and connect the different components of the design inside the selected FPGA (Altera DE2 [6] with cyclone EP2C35F672C6 in our case). With the obtained data, it is possible to be know the total amount of resources used by the design, as well as the internal delays that are very important when it is desired to make a timing simulation (essential when it is desired that the circuit continues working when it operates to high frequencies of clock). These tools allow to experienced users as well as beginners to make the process before mentioned.

3.1 Set the experiment

From 2, we take $n = 2$ and we select the case of learning the AND-gate as shown in Table 1.

Apply the learning laws in (1) to this training set and using as an initial conditions $W_0 = (0, 0)$, $b_0 = 1$, we obtain the following values:

$$W = (2, 2) \quad b = -1$$

Now we describe the main parts of the VHDL code that was used to implement the hebbian neural network.

Entity For educational purposes we design an entity that allows to the student to manipulate different signals:

1. Clock signal: The student generate this signal to check the transitory values of the weights and bias of the neural network.
2. Reset signal: The student set the initial values for the weights and bias.
3. Aprender signal: The student can set the neural network in learning and testing mode.

As an output ports we manage four seven segment displays to show values of the parameters of the network during learning or the testing steps.

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL; use
IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_SIGNED.ALL; use
work.pack_red_heb_and.ALL;


entity red_heb_and is
    Port (
              clk       : in std_logic;
              reset     : in std_logic;
              aprender  : in std_logic;
              disp_0_w1 : out std_logic_vector(6 downto 0);
              disp_1_w1 : out std_logic_vector(6 downto 0);
              disp_0_w2 : out std_logic_vector(6 downto 0);
              disp_1_w2 : out std_logic_vector(6 downto 0);
              disp_0_b  : out std_logic_vector(6 downto 0);
              disp_1_b  : out std_logic_vector(6 downto 0);
              disp_0_y  : out std_logic_vector(6 downto 0);
              disp_1_y  : out std_logic_vector(6 downto 0)
        );
end red_heb_and;
```

Architecture This fragment of code shows the implementation of hebbian neural network in two ways: the learning and the testing modes.

```
architecture una_neurona of red_heb_and is
begin
process(clk, reset)
begin
    if reset='1' then
        w1 <= "0010";
        w2 <= "0011";
        b  <= "0001";
        y  <= "0000";
        elsif (clk'event and clk='1') then
            if aprender='1' then
                w1 <= w1 + rom_x1 * rom_y;
                w2 <= w2 + rom_x2 * rom_y;
                b  <= b  + rom_y;
            else
                I <= rom_x1 * w1 + rom_x2 * w2 + b;
                    if I < 0 then
                        y <= "1111";
                    else
                        y <= "0001";
                    end if;
            end if;
```

```
    end if;
end process;
end una_neurona;
```

Since this kind of neural network uses bimodal format (+1,-1) we use the two's complement representation for the output of the seven-segment displays.

3.2 Simulations

Now we present the simulation results of the VHDL code in figure 2. Here both phases are reviewed, first the learning signal (aprender) is activated, so we can see how the transitory values of weights w_1, w_2 and bias b change in,

`disp_0_w1, disp_1_w1, disp_0_w2, disp_1_w2, disp_0_b, disp_1_b`

Then, the testing mode starts where these values are fixed and we send other input values to check the response of the neural network y_{net} in,

`disp_0_y, disp_1_y`

Due to the response is correct, we successfully finish the simulation and we proceed to download this program to the FPGA.

For this implementation we use 53 logic elements, 59 pins and 4 embedded multipliers, that is $<$ 1 percent of the total resources of the Altera Cyclone FPGA.

4 Conclusion

From an educational point of view, the design and simulation of neuronal networks as well as its implementation in programmable logic devices constitute an important element in formation of students related to design of digital circuits, electronics as well as digital control and related areas. Although commercial designs exists from several companies, its use generates technological dependency, therefore to provide a description with Hardware Description Languages such as VHDL in this case allows to have an open technology which does not have dependency disadvantages and even allows the access to all internal signals and full modification of the structure. Students of graduate and undergraduate courses can take advantage of this contribution to improve their abilities. After the simulations and the implementations, the design proved to be very efficient for demonstrations and practices in laboratory.

Acknowledgments

Dr. M. A. Moreno-Armendariz thanks to the Centro de Investigación en Computación of the IPN (CIC-IPN) and the Secretaría de Investigación y Posgrado of the IPN, under Research Grant no. 20070612 and CONACyT. Also, the authors thanks to Altera [6] for the donation of the Altera DE2 kits and academic licenses of Quartus II software.

References

1. Romero-Troncoso, R. de J., Ordaz-Moreno A., Vite-Frias J.A., Garcia-Perez A., 8-bit CISC Microprocessor Core for Teaching Applications in the Digital Systems Laboratory, Reconfig 06 (IEEE), (2006)
2. Fu L., Neural Networks in Computer Intelligence, McGraw-Hill, (1994) pp. 55-58
3. Pardo F., Boluda J.A., VHDL Lenguaje para síntesis y modelado de circuitos, Alfaomega, 2da. edición,(2004) pp. 273-309
4. Sheu B.J., Choi J., Neural information processing and VLSI, Kluwer Academic Publishers,(1995) pp. 21
5. (2007) Altera: Quartus II. [Online]. Available: http://www.altera.com/education/demonstrations/online/design-software/onl-design-software-demos.html
6. (2007) Altera: University Program. [Online]. Available: http://www.altera.com/education/univ/unv-index.html

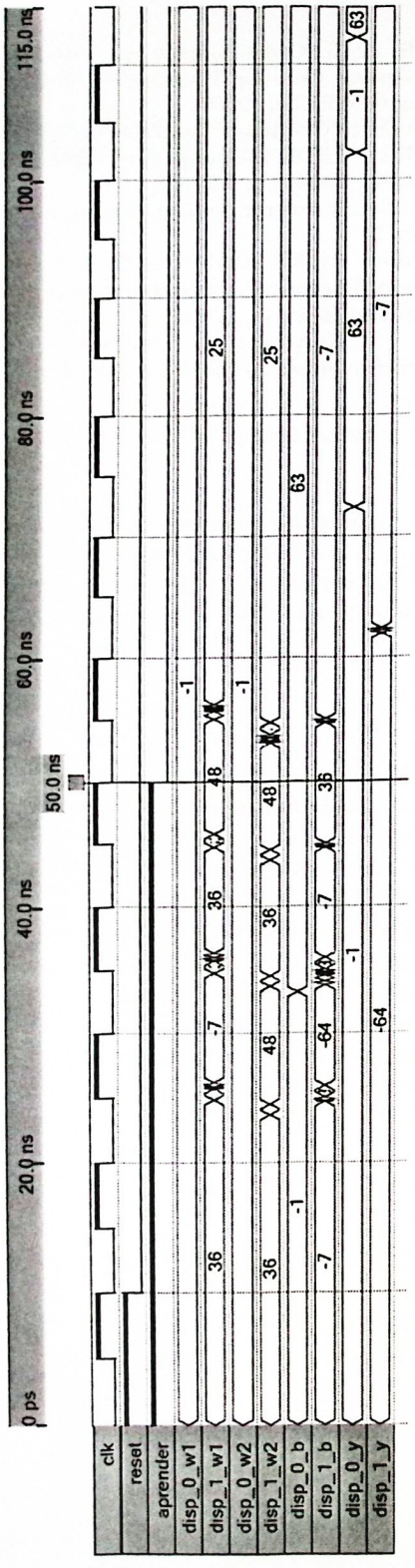

Fig. 2. Simulation waveform of the hebbian neural network.