Codesign of Fully Parallel Neural Network for a Classification Problem

Rolf F. Molz[1], Paulo M. Engel[1], Fernando G. Moraes[2], Lionel Torres[3], Michel Robert[3]

[1]Inst. de Informática – UFRGS - Caixa Postal 15064 - POA - RS. CEP: 91501-970.
[2]Fac. de Informática – PUC - Av. Ipiranga, 6681 - prédio 30 / bloco 4 - POA - RS. CEP: 90619-900
[3]LIRMM -Université Montpellier II - 161, rue Ada 34392 MONTPELLIER Cedex 5 France
E-mails: rolf@dinf.unisc.br, engel@inf.ufrgs.br, moraes@inf.pucrs.br, torres@lirmm.fr, robert@lirmm.fr

Abstract

This paper presents a neural network implementation using reconfigurable devices (FPGA) and a signal processor (DSP) available in a flexible codesign platform. The network is described using C and VHDL languages, for the software and hardware parts respectively. The software part is responsible for the learning phase and the hardware part is responsible for the propagation phase. Our objective for the hardware part is twofold: define a codification for the numbers used by the network, aiming hardware cost reduction, and exploit the inherent parallelism of neural networks, aiming speed-up the image processing. We present a synchronous implementation for the network and a codification analysis using PSNR (Peak Signal Noise Ratio).

Keywords: Neural Network, Hardware Implementation, Reconfigurable Devices, Signal Processor.

1. Introduction

Neural networks have been used broadly in many fields, either for development or for application. They can be used to solve a great variety of problems that are difficult to be resolved by other methods. Neural networks are especially useful for problems in which generalization plays an important role. Examples of applications extend from commercial to research areas.

Although, neural networks have been implemented mostly in software, hardware versions are gaining importance. Software versions have the advantage of being easy to implement, but with poor performance. Hardware versions are generally more difficult and time consuming to implement, but with better performance than software versions.

For an ANN (Artificial Neural Networks) implementation, either in software or hardware, some basic constraints must be considered:

- The network topology (feed-forward or feed-back);
- The number of inputs and outputs;
- The number of neurons;
- The number of synaptics per neuron;
- The number of layers;
- The learning algorithm;
- Operations in floating point.

For hardware implementation, a new set constraint must also be considered:

- The technology employed (analog, digital or hybrid);
- The numeric representation for inputs, synaptics weights, and outputs of activation function (fixed or floating point);
- The precision for inputs, synaptics weights, and outputs of the activation function (number of bits).

The employed technology defines the cost, performance and reconfiguration of the ANN: i) analog implementation has a good performance and low cost, but it is difficult to implement due to required references voltages [1]; ii) digital implementation (ASIC) has higher cost and poorer performance than compared to the analog one, but it is faster to implement.

Both approaches (analog and digital-ASIC) have fixed topology, resulting in an ANN suited only for one type of target application. Modern reconfigurable (FPGA) devices can be used to implement neural networks in hardware ([2][4][5][7][8]). Furthermore, digital implementation, using FPGAs, allows the redefinition of the topology using the same hardware. The disadvantage of an implementation using FPGA over ASIC is the performance. FPGAs normally run slower than ASICs.

When considering the learning phase, good precision is required to obtain correct synaptics weights. If small precision is used, then ANN does not converge and incorrect outputs will be generated during the propagation phase. The cost function of the propagation phase is speed. This is needed due the larger number of multiply-accumulate operations that must be executed. The precision during the propagation phase can be degraded due to synaptics weights being calculated for the target precision during the learning phase. The resulting errors induced for this approximation can be neglected [2].

As mentioned in the previous paragraphs, codesign techniques are well suited to implement ANNs. The implementation of floating-point arithmetic (much used during the learning phase) in FPGAs is too expensive, making its implementation prohibitive. Therefore, learning phase is a good candidate to be implemented in software (general-purpose processor - GPP). In software, floating-point representation can be used, allowing in this way convergence and accuracy of the network. On the other hand, the propagation phase must be implemented in hardware since speed is required. Using dedicated hardware, several multiply-accumulate operations can be executed in parallel. In the software implementation this parallelism is impossible, resulting in sequential execution.

This article presents an efficient hardware implementation of artificial neural networks using reconfigurable devices and a GPP, available on a flexible codesign platform – LIRMM [3]. Section 2 presents this platform. Section 3 and 4 present the neural network model and its implementation, respectively. Section 5 presents our results and Section 6 our conclusions.

2. Codesign Platform

This work use a platform developed at the Laboratory of Computer Science, Robotics and Microelectronics of Montpellier, called LIRMM (Logic Inside Reconfigurable Micro Machine) [3]. This platform was adopted because it facilitates the use of codesign techniques.

The LIRMM platform was developed in order to provide a fast prototyping environment. It is based on a signal processor (TMS 320C40 Texas) to implement the software part of a given application, and two reconfigurable devices (XC4013E FPGAs - Xilinx) to implement the hardware part of the same application. A bus with 34 bits interconnects the FPGAs, facilitating hardware partitioning. Additionally, there is a local static RAM (128 Kbytes) for each FPGA.

The software part is described using the C programming language, and the hardware part is described using VHDL (Figure 1). The FPGA is configured using CAD tools for logical synthesis, technology mapping, placement, routing and download.

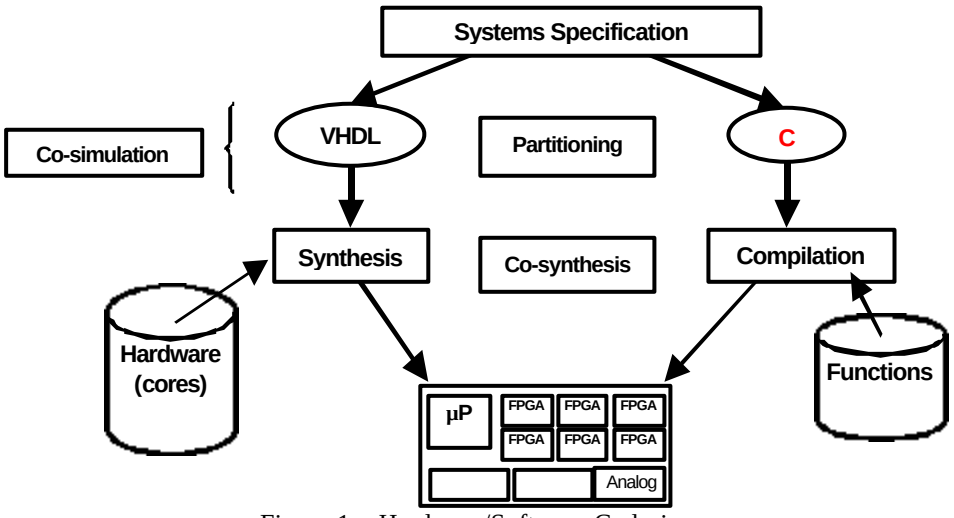


Figure 1 – Hardware/Software Codesign

3. Neural Network Model

In this section, we describe the technique used to implement the MLP (Multi-Layer Perceptron) ANNs with backpropagation learning. Such network was implemented on the platform discussed in the previous section.

The multiply-accumulate operation is required to calculate the net value of each neuron (Equation 1). This operation is the bottleneck when implementing ANNs into FPGAs, since multiply-accumulate operations require a large amount of logic devices. After the net value was calculated, the neuron output value is obtained through an activation function. For a feedforward neural network with backpropagation learning, a sigmoid function can be used (Figure 2).

$$net_i = \sum_{j=0}^{n} w_{ij} * I_j \qquad (1)$$

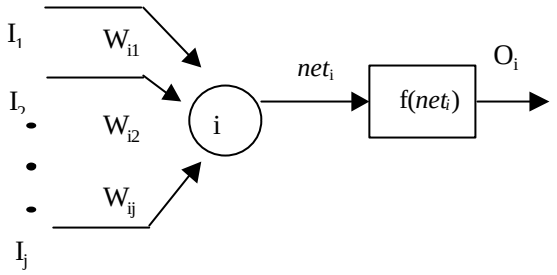


Figure 2 – Neural Network Model

Simple additions or subtractions must replace the multiply-accumulate operation in order to reduce the size of the resulting circuit. The following methods may be used:

- Serial multiplication, in which only one multiplier is implemented [6], resulting in a small circuit, but with poor performance. In this method, the input patterns have to be serialized to accomplish the multiplication at bit level, and not at word level.
- Distributed arithmetic rule. This implementation uses the FPGAs internal structure, i.e., the lookup tables [7]. The disadvantage of such method is that one of the numbers to be multiplied must be constant, making the implementation of the learning phase in the FPGA impossible.
- Shifting operations [8]. This method requires numbers to be represented as power of two[1]. It has a good performance, and allows the learning phase to be implemented in the FPGA. The drawback of such method is to find adequate numbers such that the output error is minimized [2].

In this paper, multiplication is implemented with shifting operations followed by partial sums. The objective is to achieve convergence with minimum error and to implement the learning phase in the FPGA.

In [8], asymmetric codification (power of two) of the input patterns (I_j in Equation 1) and of the activation function induces to errors at the output of the neuron. In order to solve this problem we have adopted a different type of codification. Numbers are represented in a 4-bit fix-point format (Equation 2), where 1 bit is used for sign (s) and three bits are used for the fractional part (f).

$$\text{number} = \begin{cases} -1^s * 0.f & \text{for } f \neq 0 \\ -1^s & \text{for } f = 0 \end{cases} \qquad (2)$$

The values of the synaptics weights (w_j in Equation 1) are represented by a 10-bit fixed-point number, where 1 bit is used to represent the sign, 4 bits are used to represent the integer part, and 5 bits are used to represent the decimal part (defined through simulations). This codification can represent values from 15,96875 to -15,96875. It is important to choose the correct range of weights in order to avoid neuron output saturation.

[1] This causes the intervals between numbers to be asymmetric, e.g., 0.5, 0.25, 0.125, 0.0625.

Although threshold functions are easier to implement, they cannot be used in pattern recognition applications. Sigmoid function can be implement as a piecewise function, with discrete values fixed in function of the net value [2]. The activation function in this work is implemented by means of lookup tables (using the internal structure of FPGAs). The resulting curve of the sigmoid functions is presented in Figure 3.

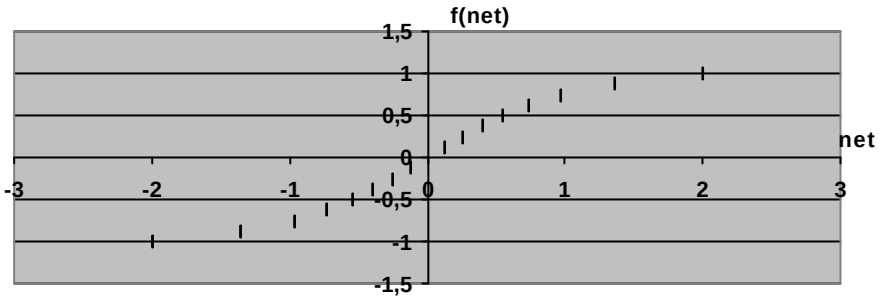


Figure 3 – Activation function

4. Neural Network Implementation

A 3-layer ANN is implemented with two neurons in the input layer, three neurons in the hidden layer, and one neuron in the output layer. The function of the input layer is to adjust the input values to the internal neural codification (hardware codification). Considering that the input signals have already been coded in the DSP, this layer is not implemented. The learning process adopted is the backpropagation algorithm.

The hardware/software partitioning is defined in function of delay and area constraints. As mentioned in the previous section, the hardware part is responsible for propagation of the input patterns. The software part is responsible for updating the synaptics weights and communication with the computer host (responsible for I/O operations).

The learning phase is coded in C-language, taking into account the hardware codification. When new weights are obtained in the learning phase, they are automatically converted to the hardware codification. This process can increase CPU usage for the learning process. However, it will reduce errors induced by the hardware codification.

Figure 4 shows a block diagram of the ANN part implemented in hardware. In the figure, the hidden layer (neurons connected to the inputs) and the output layer are shown. We have adopted a synchronous implementation, in which each layer has clock cycles to compute its output, as in pipeline architecture. The number of clock cycles is obtained in function of the neuron delay, obtained from a time analysis tool. This results in a stable network, since correct values are guaranteed at the neuron outputs.

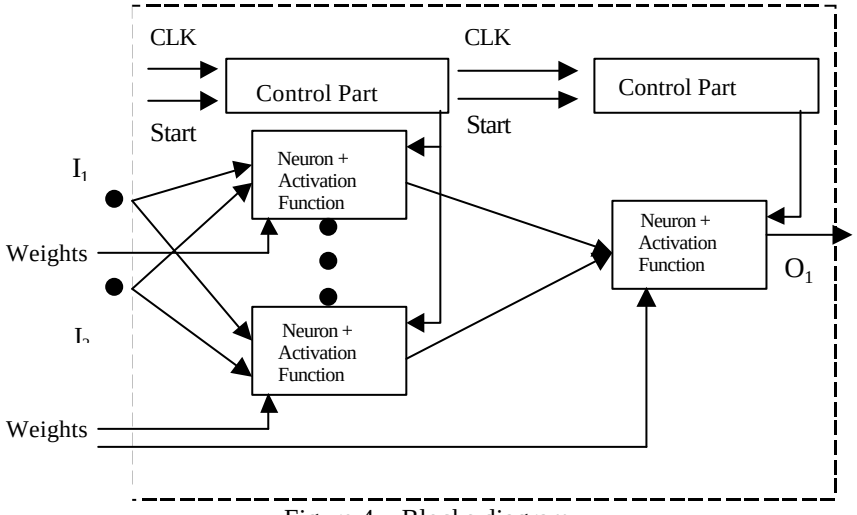


Figure 4 – Blocks diagram

Figure 5 presents the pseudo-code for the software/hardware interaction during the learning phase. The software part generates random synaptics weights (step 2), sends these synaptics weights as a set of inputs to the hardware part (step 3). After propagation, the hardware part sends the output to the processor (step 4), which adjusts the synaptics weights (step 6). This process is repeated until a minimum error condition is achieved.

1)Software :	Read the patterns and desired output of the neural network (stored in hard disk)
2)Software :	Random generation of synaptic weights
3)Software :	Send the synaptics weights to FPGA
4)**Hardware**:	Propagation of the pattern through the neural network / sending the output
5)Software :	Read the result of the FPGA
6)Software :	Adjust the synaptic weights
7)Software:	Error calculation and verification of output condition
8)	Go to Step 3

Figure 5 - Software/hardware interaction in the learning phase

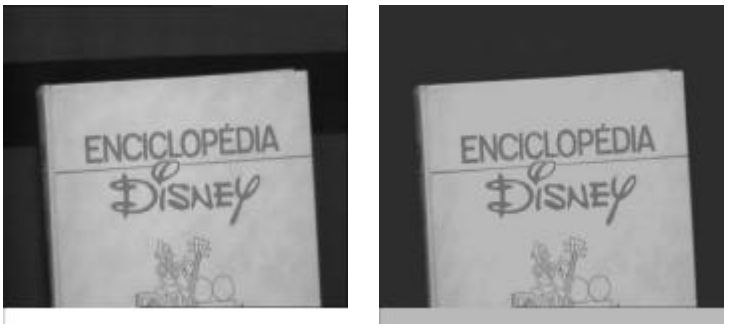
Figure 6 presents the pseudo-code for the software/hardware interaction during the propagation phase. The process is analogous to the learning phase, with no interactions, since synaptics weights are calculated in the learning phase.

1)Software:	Read the desired patterns
2)Software :	Send to FPGA the synaptics weights
3)**Hardware**:	Propagation of the pattern through the neural network
4)Software :	Read the result of the FPGA

Figure 6 - Software/hardware interaction in the propagation phase

5. Results

A typical application in the processing image area was used as a validation benchmark. The task consists in locating letters of a giving image. Figure 7.a shows an image to be processed using ANN. This image is obtained by a CCD-camera, with a resolution of 512 x 512 pixels and 256 gray-scale level.



(a) – Original Image (b) – Clusterized Image in 16 colors

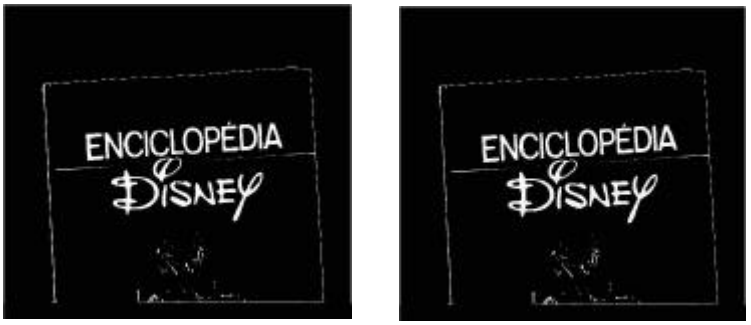
Figure 7 – Original image and it clusterized image

The input patterns are the pixels of the original image and the average of neighbor pixels. This average is obtained in a 3x3-pixel window, centered on the pixel.

As explained in Section 3, input codification allows only 16 different values (1 bit for sign and 3 bits for value). Therefore, the original image, coded in 256 gray levels, need to be pre-processed to reduce the gray levels from 256 to 16. Different clustering methods can be used for pre-processing; e.g., uniform quantization, area growing or Kohonen Maps. We have

implemented the Kohonen Maps to pre-process the original image. Figure 7.b shows the clusterized image using 16 gray values.

The pre-processed image, with each pixel coded in 16 gray-levels, is the start image to be treated by the ANN. The learning phase is accomplished by the DSP, i.e., in software. After the learning phase, the synaptics weights are send to the FPGA. The whole image is processed with all neurons, in each layer, working in parallel. The result is showed in Figure 8.a. To compare the obtained result (Figure 8.a), the propagation phase is also performed by the DSP (Figure 8.b).



(a) Classified image by hardware (5 bits for the factional part)

(b) Classified image by software (complete floating point representation)

Figure 8 – Classified images by hardware and software

Figure 9 shows the histogram resulting from the difference between the classified images (Figure 8.a and Figure 8.b). As can be observed, the image processed by hardware is very close of the one processed by software, since the difference between almost all pixels is zero.

Figure 9 – Histogram comparing the classified images

To quantify this difference a PSNR (Peak Signal Noise Ratio) analysis is performed. The result is expressed in dB, being proportional to the similarity between images. The goal of this analysis is to verify the error induced by the codification of the synaptics weights, in function of the number of bits in the fractional part. The PSNR is obtained by Equation 3:

$$output = 10 \log \left(\frac{signal}{error} \right) \qquad (3)$$

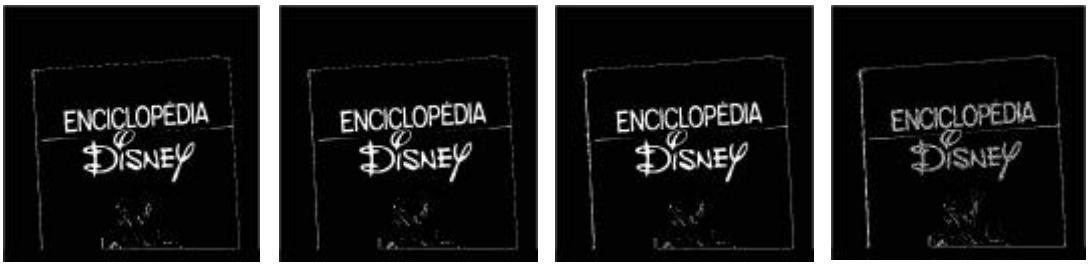
Where: signal = number of pixels multiplied by the square of the larger pixel found in the classified image by software.
error = sum of the squares of all differences between corresponding elements of the two input images.

Table 1 shows PSNR values for different number of bits in the fractional part. The PSNR gain is not proportional to the number of bits in the fractional part. A good tradeoff between area (number of bits) and gain is achieved with 5 bits, proving the correctness of our codification.

Table 1 –PSNR Results

Number of the bits for the fractional part	PSNR (dB)
3	18.0424
4	25.4324
5	31.0139
8	36.5052
9	36.6598

Figure 10 shows the images processed with different number of bits in the fractional part. As expected, a codification with 3 and 4 bits result in a poor classification.



(a) 9 bits (b) 8 bits (c) 4 bits (d) 3 bits

Figure 10 – Images classified by hardware

6. Conclusions

This paper has presented the implementation of neural networks by reconfigurable devices and signal processor using the LIRMM platform. A fully parallel neural network implementation was adopted, since our goal was to exploit the inherent parallelism of neural networks.

The neural network description is modular, being possible to easily increase or decrease the number of neurons. The description in the synchronous mode allowed us to exploit this modularity and to maintain the accuracy of the results of the neural network.

Two shift-registers were used to implement the control part of the neural network, each one with a delay of 10 clock periods. As the frequency of the FPGA is 20 MHz, 1 μs is necessary to propagate the input patterns. Considering the DSP frequency, 50 MHz, 1 μs will result in 50 clock cycles. We have not yet quantitatively compared hardware and software performances, but these data indicate that the hardware implementation has a better performance than the software one, once that the DSP is not able to execute all the propagation tasks in just 50 cycles.

One of our contributions is the definition of the number of bits required to code the synaptics weights, using a PSNR analysis. The number of bits plays a major rule in the hardware cost, since the area necessary to implement multiply-accumulate operations is proportional to the square of the number of bits. Therefore, the number of 5 bits for the fractional part represents a good tradeoff, allowing the ANN implementation in a single FPGA (XC-4013).

Acknowledgements

The author Rolf Fredi Molz would like to gratefully acknowledge the support of CAPES Capes/Cofecub project number 245/98-Brazil.
The author Fernando G. Moraes would like to gratefully acknowledge the support of CNPq project number 522939/96-1-Brazil.

Reference

[1]MOLZ, Rolf F; ENGEL, P.M. "A New Proposal for Implementation of Competitive Neural Networks in Analog Hardware". *Vth Brazilian Symposium on Neural Networks*, Belo Horizonte – MG, pp.186-191. December, 1998.

[2]MOLZ, R. F.; ENGEL, P.M. ; MORAES, F.G. “Uso de um ambiente codesign para a implementação de redes neurais”,

IV Congresso Brasileiro de Redes Neurais, p. 13-18, São José dos Campos, Brasil, July 1999.

[3]TORRES,L.; PILLEMENT,S.; ROBERT,M.: “LIRMM: Prototyping Platform for Hardware/Software Codesign”, *Journal of Current Issue in Electronic Modeling*, Issue 9, p. 49-59, March 1997.

[4]FIGUEIREDO M., GLOSTER C. "Implementation of a Probabilistic Neural Network for Multi-Spectral Image Classification on a FPGA CustomComputing Machine". *Vth Brazilian Symposium on Neural Networks* - Belo Horizonte, Brazil, December 9-11, 1998.

[5]FIGUEIREDO M. A., FLATLEY T. P., STAKEM P. H. , HINES T. M. "Extending NASA's Data Processing to Spacecraft". *IEEE Computer* Volume 32 number 6, pages 115-118, June 1999.

[6]GSCHWIND, M.; VALENTINA, S.; MAISCHBERGER, O. "Space Efficient Neural Net Implementation". *Proc. of the Second International ACM/SIGDA Workshop on Field Programmable Gate Arrays*. Berkeley, CA. Febr., 1994.

[7]ROSADO, Alfredo; BATALLER, M.; SORIA, E.; CALPE, J.; FRANCÉS, J.V. "A new hardware architecture for a general-purpose neuron based on distributed arithmetic and implemented on FPGA devices". 321-326p. 1998.

[8]CLOUTIER, Jocelyn; SIMARD, P.Y. "Hardware Implementation of the Backpropagation without Multiplication". *Proc. of the IV International on Microelectronics for Neural Networks and Fuzzy Systems*, 46-55p. Turim, Italy. Sept. 1994.