Available online at www.sciencedirect.com
ScienceDirect
Microprocessors and Microsystems 30 (2006) 435–444
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

Hardware description of multi-layer perceptrons with different abstraction levels

E.M. Ortigosa [a,*], A. Cañas [a], E. Ros [a], P.M. Ortigosa [b], S. Mota [a], J. Díaz [a]

[a] Department of Computer Architecture and Technology, ETS Ingeniería Informática, University of Granada, E-18071 Granada, Spain
[b] Department of Computer Architecture and Electronics, University of Almería, E-04120 Almería, Spain

Available online 27 April 2006

Abstract

This paper presents different hardware implementations of a multi-layer perceptron (MLP) for speech recognition. When defining the designs, we have used two different abstraction levels: a register transfer level and a higher algorithmic-like level. The implementations have been developed and tested into reconfigurable hardware (FPGA) for embedded systems. We also present a comparative study of the costs for the two considered approaches with regards to the silicon area, speed and required computational resources. The research is completed with the study of different implementation versions with diverse degrees of parallelism. The final aim is the comparison of the methodologies applied in the two abstraction levels for designing hardware MLP's or similar computational structures.
© 2006 Elsevier B.V. All rights reserved.

Keywords: FPGA; Hardware implementation; Multi-layer perceptron; Speech recognition

1. Introduction

An Artificial Neural Network (ANN) is an information processing paradigm which mimics the way biological nervous systems process information. An ANN is configured through a learning process for a specific application, such as pattern recognition or data classification, and as in all biological systems, this learning process will require the adjustment of the synaptic connections between the neurons.

ANNs implemented in software are becoming more and more appropriate for real-world applications [1,2]: Optical Character Recognition (OCR), data mining, image compression, medical diagnosis, Automatic Speech Recognition (ASR), etc. Currently, ANN hardware implementations run in a few niche areas [3]: where very high performance is required (e.g., high-energy physics); in embedded applications of simple hardwired networks (e.g., speech recognition chips); and in neuromorphic systems which directly implement a desired function (e.g., touchpad and silicon retinas).

We studied the viability of implementation and efficiency of ANNs into reconfigurable hardware (FPGA) for embedded systems, such as portable real-time ASR systems for consumer applications, vehicle equipment (GPS navigator interface), toys, aids for the disabled, etc. Among the different ANN models used for ASR, we focused on hardware implementation of a multi-layer perceptron (MLP). Recently, different authors have implemented several models of ANNs on FPGA devices spurred on by the fast advances in this kind of technology [4–8].

We concentrated on *a voice controlled phone dial system* (this may be of interest to drivers to avoid distraction while driving). The chosen application provided us with the MLP parameters to be implemented and the word set of interest (numbers from 0 to 9).

In recent years, FPGA technology has made very significant advances, enabling the implementation of highly complex systems. The most widely used description languages are VHDL and Verilog, but other higher-level

* Corresponding author. Tel.: +34 958240460; fax: +34 958243226.
E-mail address: eva@atc.ugr.es (E.M. Ortigosa).

0141-9331/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2006.03.004

description languages such as System-C and Handel-C are being developed and currently constitute a valid alternative for fast designs [9].

In this paper, we describe the design of the system at two abstraction levels: a register transfer level (RTL) and a higher algorithm-like level. As results depend on the designer's skills, a comparison of the implementations of the system is seldom addressed [10,11]. The implementations have been defined using VHDL and Handel-C, respectively. On the other hand, these types of comparisons are of great interest to designers who have to choose a hardware description language for a system of great complexity. Hardware resource consumption and design time are important factors to be taken into consideration. This contribution also discusses these different factors in relation to the design of a system for a real-world problem and evaluates the pros and cons of the different approaches presented.

The paper is organized as follows. Section 2 introduces the MLP structure adapted for speech recognition applications. In Section 3, both serial and parallel versions of the MLP are described using two different abstraction levels: a register transfer level and a higher algorithmic level. These implementations were defined using standard VHDL, as the hardware description language, and Handel-C, as a system-level specification language. The results are summarized in Section 4, followed by the conclusions.

2. MLP and speech recognition

Automatic speech recognition is the process by which a computer maps an acoustic speech signal to text [12]. Typically, speech recognition starts with the digital sampling of speech. The raw sampled waveform is not suited as direct input for a recognition system. The commonly adopted approach is to convert the sampled waveform into a sequence of feature vectors using techniques, such as filter bank analysis and linear prediction analysis. The next stage is the recognition of phonemes, groups of phonemes, or words. This last stage is achieved in this work by Artificial Neural Networks (ANNs) [13]. Other techniques can be used, such as Dynamic Time Warping (DTW) [14], Hidden Markov Models (HMMs) [14], expert systems or combinations of them.

The most widely used neural classifier is the MLP, which has also been extensively analyzed and for which many learning algorithms have been developed [15].

The multi-layer perceptron neural network model consists in a network of processing elements or nodes arranged in layers. Typically, it requires three or more layers of processing nodes: an input layer which accepts the input variables used in the classification procedure, one or more hidden layers, and an output layer with one node per class (Fig. 1a). The principle of the network is as follows: data from an input pattern are presented at the input layer and the network nodes perform calculations in the successive layers until an output value is computed at each of the

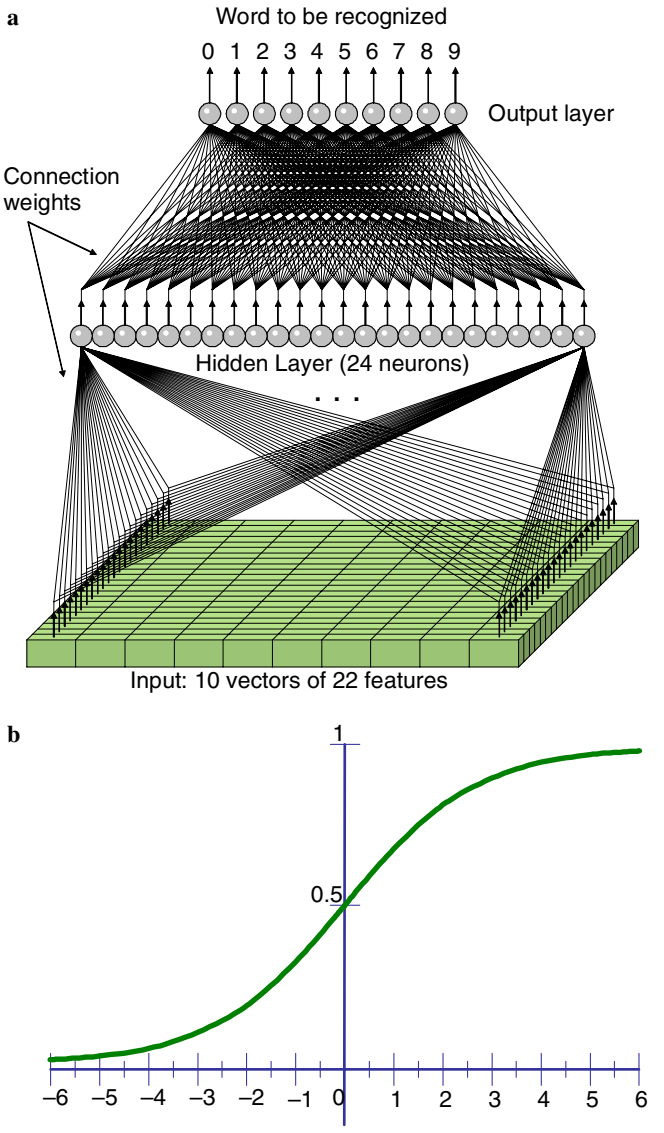

Fig. 1. (a) Example of the MLP for isolated word recognition; (b) most significant interval of the sigmoid activation function.

output nodes. This output signal should indicate the appropriate class for the input data, i.e., we expect to have a high output value on the correct class node and a low output value on the rest of the nodes.

Every processing node in one particular layer is usually connected to every node in the layers above and below it. The weighted connections define the behaviour of the network and are adjusted during training through a supervised training algorithm called backpropagation [15].

In the "forward pass", an input pattern vector is presented to the input layer. For successive layers, the input to each node is the sum of the scalar products of the incoming vector components with their respective weights:

$$\mathrm{sum}_i = \sum_j w_{ij}\mathrm{out}_j, \tag{1}$$

where w_{ij} is the weight connecting node j to node i and out_j is the output from node j.

The output of a node i is $\text{out}_i = f(\text{sum}_i)$, which is then sent to all nodes in the next layer. This continues through all the layers of the network until the output layer is reached and the output vector is computed. The function f denotes the activation function of each node. A sigmoid activation function is frequently used Eq. (2). The most significant interval of the sigmoid curve is illustrated in Fig. 1b.

$$f(\text{sum}_i) = \frac{1}{1 + e^{-\text{sum}_i}}. \quad (2)$$

3. Hardware implementations

In practice, there are a lot of applications which require embedded processing in portable devices, and which are restricted by very constraining features, such as low cost, low power, and reduced physical size. We have implemented a MLP-based system in hardware as speaker-independent isolated word recognition platform.

For our test bed application, we chose an MLP with 220 data inputs (10 vectors of 22 features extracted from speech analysis), and 10 output nodes in the output layer (corresponding to the 10 recognizable words). After testing different architectures, the best results (96.83% correct classification) were obtained with 24 nodes in the hidden layer (Fig. 1a).

For the MLP implementation, we chose fixed point computations with two's complement representation and different bit depths for the stored data (inputs, weights, outputs, activation function, etc). It is necessary to limit the range of different variables:

- Inputs to the MLP and output of the activation function: both of them must have the same range to easily manage multiple-layers processing (we chose 8 bits).
- Weights (8 bits).
- Inputs to the activation function: defined by a Look-Up-Table (LUT) storing the useful values. Apparently, we need 23 bits for the input of this function, but if we observe the sigmoid waveform (Fig. 1b), most of the values are repeated and only a small transition zone (≈1% of values) needs to be stored.

After adopting all these discretization simplifications, our model stills achieves similar classification results. For instance, in a phoneme recognition application (not word recognition), we obtained 69.33% of correct classifications with a continuous model and 69.00% using the corresponding discrete model. The results of the hardware system differ in less than 1% from the software full resolution results.

The main storage strategy is to use RAM modules so that the inputs, the outputs, and the associated weights will be stored in these RAM modules.

In the following subsections, both a serial and a parallel version of the MLP architecture are described at two different abstraction levels.

3.1. Register transfer level

The register transfer level design of the MLP was implemented using standard VHDL as the hardware description language. Though this language allows three different description levels, we implemented both serial and parallel architectures using RTL in order to compare them with those obtained with a higher algorithmic-like level.

All the design processes have been carried out using FPGA Advantage 5.3 tool, from Mentor Graphics [16], a robust HDL-based methodology for FPGA design.

In order to fix the architecture for the MLP described in Section 2, an analysis of the kind of operations and information processes is required. As can be seen in Eq. (1), the basic computations of a single neuron are the multiplication of the outputs from the connected neurons by their associated weights, and the summation of these multiplied terms. Fig. 2 describes the basic structure of the functional unit (processing element) which performs these computations. It comprises an 8-bit multiplier and a 23-bit accumulative adder. In order to connect the 16-bit multiplier output to the 23-bit adder input, a sign extension unit had to be introduced. Note that 23 bits are needed to accumulate the multiplication 220 inputs with their weights.

3.1.1. Serial version

The serial architecture has been designed to estimate the minimum area required to implement the MLP, although this implies a large execution time. Therefore, this serial version of the MLP consists in a single functional unit that carries out all the computations for every the neuron as shown in Fig. 3.

The inputs of the functional unit are both the synaptic signals and their associated weights, which have been

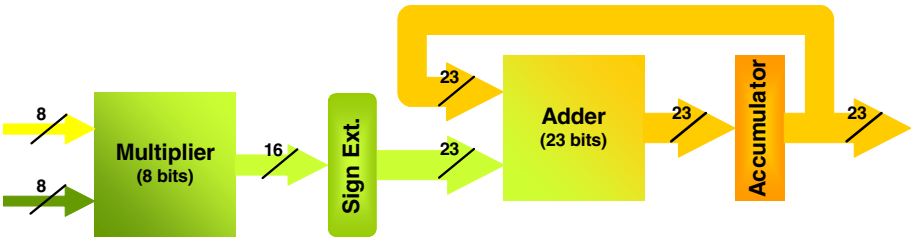

Fig. 2. Basic structure of the functional unit.

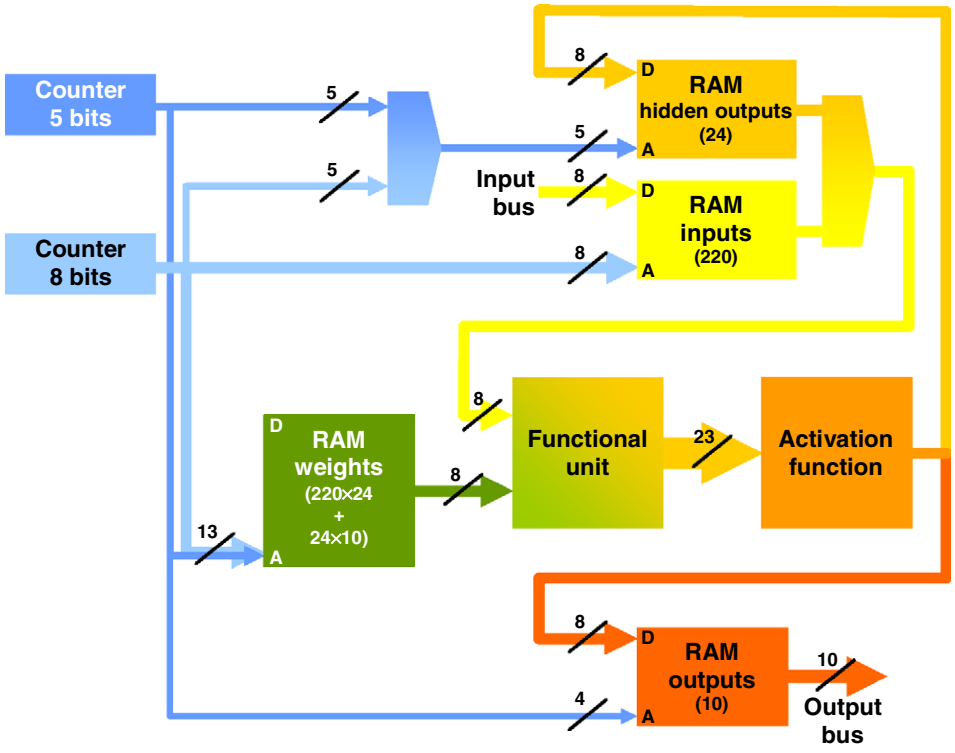

Fig. 3. Structure of the serial MLP.

stored in separate RAM modules. There is a single RAM module to store all the weights and two separate ones to store the synaptic signals: one for the values of the input layer and another for the values of the hidden neurons. By separating the RAM modules, the MLP will be able to read from the module associated with input layer and to write the output of the hidden neurons in the same clock cycle.

The output of the functional unit is connected to the activation function module. The activation function output is stored either in the hidden RAM or in the final neuron RAM, depending on the layer of the computed neuron.

The addressing RAM has been carried out by both 8-bit and 5-bit counters. The 8-bit counter addresses the synaptic signal RAM modules when reading them, and the 5-bit counter addresses them when memory writing.

The 13-bit address of the weights RAM module is computed by merging the addresses of both the 5-bit and 8-bit counters, so that the most significant bits of the merged address correspond to those of the 5-bit counter (see Fig. 3).

The control unit associated to this data path has been designed as a finite–state machine, implementing the detailed description of all necessary states, their outputs and transitions.

3.1.2. Parallel version

The proposed parallel architecture describes a kind of 'node parallelism', in the sense that it requires one functional unit per neuron when working at a determined layer. With this strategy, all the neurons of a layer work in parallel and therefore get their outputs simultaneously. This is not a fully parallel strategy because the outputs for different layers are obtained in a serial manner.

For our particular MLP, where 24 neurons exist at a hidden layer and 10 at the output layer, 24 functional units are required. All functional units will work in parallel when computing the outputs of the hidden layer, and only 10 of them will work when the output layer is computed. Fig. 4 shows the basic structure of this parallel version.

Given that all functional units work in parallel for each synaptic signal, they need access to the associated weights simultaneously, and hence, the weight RAM should be private for each one. For this reason, the 8-bit counter is used to address 24 RAM modules containing 220 weights (as indicated in Fig. 4).

Since the data transmission between layers is serial, every functional unit will also need some local storage for output data. We decided to use 24 parallel registers instead of the RAM module for the serial version, since this option reduces the writing time. The data in these parallel registers are introduced to the single activation unit by a 24:1 multiplexer whose select signals are the 5-bit counter outputs. The output of the activation unit is either used as a synaptic signal for the 10 neurons at the final layer, or it is stored in the output RAM module. A new control unit has been designed for this parallel data path.

3.2. High description level

The high-level design of MLP was defined using Handel-C [17] as a system-level specification language.

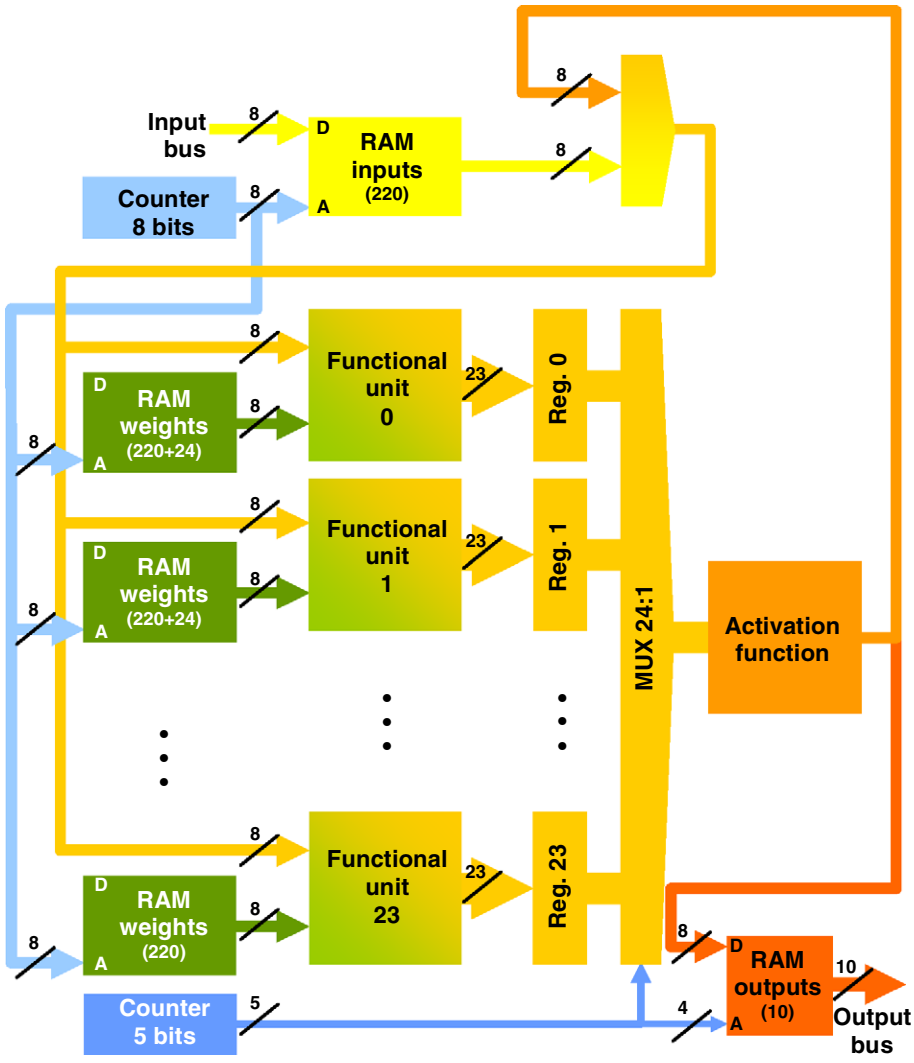

Fig. 4. Structure of the parallel MLP.

Handel-C is a language for rapidly implementing algorithms in hardware straight from a standard C-based representation. Based on ANSI-C, Handel-C includes a simple set of extensions required for hardware development, such as flexible data widths, parallel processing, and communication between parallel threads. The language also employs a simple timing model that gives designers control over pipelining without the need to add specific hardware definition to the models. This language also eliminates the need to exhaustively code finite-state machines by proving the ability to describe serial and parallel execution flows. Therefore, Handel-C provides a familiar language with formal semantics for describing system functionality and complex algorithms, resulting in a substantially shorter and more readable code than RTL-based representations.

Taking these Handel-C properties into account both the serial and parallel architectures of the MLP have been implemented in an algorithmic description level by defining their functionality as shown in the next subsections.

Although a detailed study on the possible architecture for both implementations has not been done in this present work, we have completed an analysis on how the information will be coded and stored. The designer should specify aspects like the variable widths (number of bits), the required bits for each computation, the variables that should be stored in RAM or ROM modules, the kind of storing memory [distributed by FPGA general purpose LUT resources, or using Embedded Memory Blocks (EMBs)], etc. Though Handel-C provides for quicker working hardware systems, implementation of an improved system requires the analysis of the possible options. A small modification in the Handel-C code can lead to a large change in the final architecture, and therefore to a considerable variation in the area resources or clock cycle.

The whole design process was defined using DK Design Suite tool from Celoxica [17]. Serial and parallel designs have been finally compiled using the development environment ISE Foundation 3.5i from Xilinx [18].

3.2.1. Serial version

Only the functionality is necessary in the description of the serial MLP. To enable a comparative analysis, this description should be identical to that of the RTL already described in the previous section. Thus, the MLP computes the synaptic signals for each neuron in the hidden layer by processing the inputs sequentially; and subsequently, the obtained outputs are similarly processed by the output neurons.

From an algorithmic point of view, this functional description implies the use of two different loops: one for computing the results for neurons in the hidden layer, and another for neurons at the final layer. As an example of the high-level description level reached when designing with Handel-C, the following shows the programmed code for the first loop, and its functionality corresponds to the kind of computation shown in Eq. (1).

```
for (i=0; i<NumHidden; i++)
  {
Sum = 0;
for (j=0; j<NumInput; j++)
  Sum += W[i][j]*In[j];
...
  }
```

where `NumHidden` is the number of neurons in the hidden layer (24), `NumInput` is the number of inputs (220), `W` is the array containing the weight values required for processing the inputs and computing the synaptic signals at hidden layer (24×220), `In` is the array input, and `Sum` is a variable that stores the weighted (accumulated) sum of the multiplication of the inputs by their respective weights. From the code, it can easily be seen that all multiplications are being computed sequentially.

3.2.2. Parallel version

As in the serial case, the functionality of the parallel version is similar to that of RTL description, as all neurons belonging to the same layer compute their results simultaneously, in parallel. The only exception is the access to the activation function which is carried out in a serial manner.

The parallel designs were defined through the Handel-C "par" directive, which forces the implementation of dedicated circuits, so that a defined part of an algorithm can be computed in parallel. In order to make optimal use of the "par" directive, we would need to identify the set of computations and information transfers which can be done in parallel. For example, the designer would have to make sure that RAM modules only allow one location to be accessed in a single clock cycle.

Consequently, the serial loop shown in previous subsection can be parallelized as:

```
par (i=0; i<NumHidden; i++)
  Sum[i] = 0;
par (i=0; i<NumHidden; i++)
  for (j=0; j<NumInput; j++)
    Sum[i] += W[i][j]*In[j];
```

4. Results

The systems have been designed using the development environments *FPGA advantage* for the RTL version, and *DK Design Suite* for the higher algorithmic-like version. Then the designs have been translated into EDIF files, and finally, placed and routed in a Virtex-E 2000 FPGA, using the development environment *ISE Foundation 3.5i.*

The basic building block of the Virtex-E CLB (Configurable Logic Block) is the Logic Cell (LC) [19], which comprises a 4-input function generator, carry logic, and a storage element. Each Virtex-E CLB contains four LCs organized in two similar *slices*, and logic which combines function generators to provide functions of five or six inputs. Consequently, when estimating the number of system gates provided by a given device, each CLB counts as 4.5 LCs [19]. The 4-input function generator (included in each LC) is implemented as 4-input LUTs. Each LUT can provide a 16×1 synchronous RAM. Virtex-E also incorporates large Embedded Memory Blocks (EMBs) which complement the distributed RAM memory resources available in the CLBs.

The results of each implementation can be characterized by the following parameters: number of slices, number of EMBs RAM, maximum clock rate, and data throughput (D_t) as the number of evaluated input vectors per second (note that each vector consist of 220 components). In order to make comparisons between the different implementations easier, we have also estimated the number of system gates while trying to represent the global hardware resources (H_r) and the data throughput (D_t) of each approach. Furthermore, to better illustrate the trade-off between these two characteristics, which can be adopted as a convenient design objective during the architecture definition process, we have evaluated the performance cost as $P_c = H_r/D_t$. In this way, we can evaluate how much hardware cost is required by a given performance.

For the hardware resources (H_r) estimation, the gate counting used on Virtex-E devices is consistent with the system gate counting used on the original Virtex devices [19]. The basics of the system gates counting are as follows: each logic cell used as general logic is weighted as 12 system gates, each logic cell used as distributed memory represents 64 system gates (16 bits $\times$ 4 gates/bit), and finally, the resource cost for embedded memory adds up to 4 system gates/bit.

Table 1 shows the implementation results obtained after synthesizing both serial and parallel versions of the MLP defined at a RTL abstraction level.

As expected, results indicate that the serial version, with only a functional unit, requires less area resources than the

Table 1
Implementation characteristics of the designs with VHDL

MLP design	Hardware resources			Performance			Performance cost (P_c)
	# Slices	# EMBs RAM	# System gates	Max. Freq. (MHz)	# Cycles	D_t (data/s)	# System gates/D_t
Serial	1530	0	187677	24.7	5588	4440	42.27
Parallel	3094	0	256581	20.2	278	74150	3.46

parallel one, where a functional unit per hidden neuron was included. In the parallel version, the data throughput (D_t) is 17 times higher, mostly due to the reduced number of clock cycles needed for each input vector evaluation. In this way, we are taking advantage of the inherent parallelism of the ANN computation scheme. After synthesizing the design, a slight difference can be observed in the maximum clock frequency, this is due to the different maximum combinational paths of the two approaches.

Table 2 represents the results obtained after synthesizing the serial and parallel versions of the MLP defined at an algorithmic-like description level (using Handel-C as HDL). As commented in Section 3.1.1, obtaining an efficient system requires a careful analysis of the possible choices. When defining the MLP, we had to decide how to store the data in RAM. Given the simplicity of Handel-C when defining different implementation strategies, we chose to evaluate three different versions: (a) only using distributed RAM for the whole design; (b) the weights associated to synaptic signals (large array) make use of EMBs RAM, while the remaining data are stored in a distributed mode; (c) only requiring memory grouped in EMBs RAM modules.

Results in Table 2 show that, regardless of the distribution memory option, the parallel version obviously requires more area resources than the serial one. As occurred in the VHDL system described above, the data throughput of the parallel versions is about 17 times higher than the serial approaches.

Fig. 5 shows the results obtained for the data throughput. When considering the designs at the different abstraction levels, we can observe that, although the RTL with VHDL achieves the best results, it just manages to outperform the best version described at an algorithmic-like abstraction level, (1.24 and 1.21 times faster in the serial and the parallel versions, respectively). When considering the different memory utilization strategies, the three versions achieve similar data throughputs.

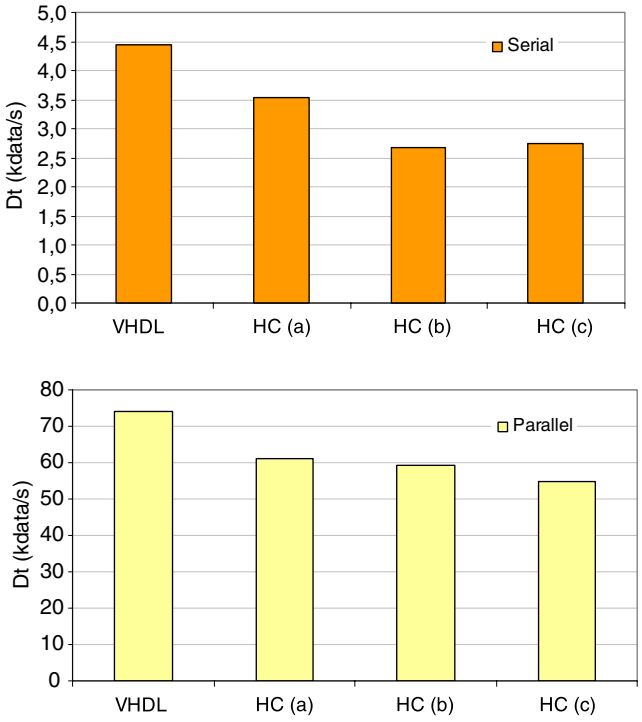

Fig. 5. Data throughput (D_t) of the different evaluated designs.

Fig. 6 shows how the performance cost (P_c) associated to each design strategy at different abstraction levels is

Table 2
Implementation characteristics of the designs with Handel-C

MLP design	Hardware resources			Performance			Performance cost (P_c)
	# Slices	# EMBs RAM	# System gates	Max. Freq. (MHz)	# Cycles	D_t (data/s)	# System gates/D_t
(a)							
Serial	2582	0	245268	19.7	5588	3535	69.38
Parallel	6321	0	333828	17.2	282	60968	5.48
(b)							
Serial	710	11	218712	16.1	5588	2671	81.86
Parallel	4411	24	518928	16.7	282	59326	8.75
(a)							
Serial	547	14	248996	15.3	5588	2733	91.07
Parallel	4270	36	695380	15.4	282	54692	12.71

(a) Only distributed RAM. (b) Both EMBs RAM and distributed RAM. (c) Only EMBs RAM.

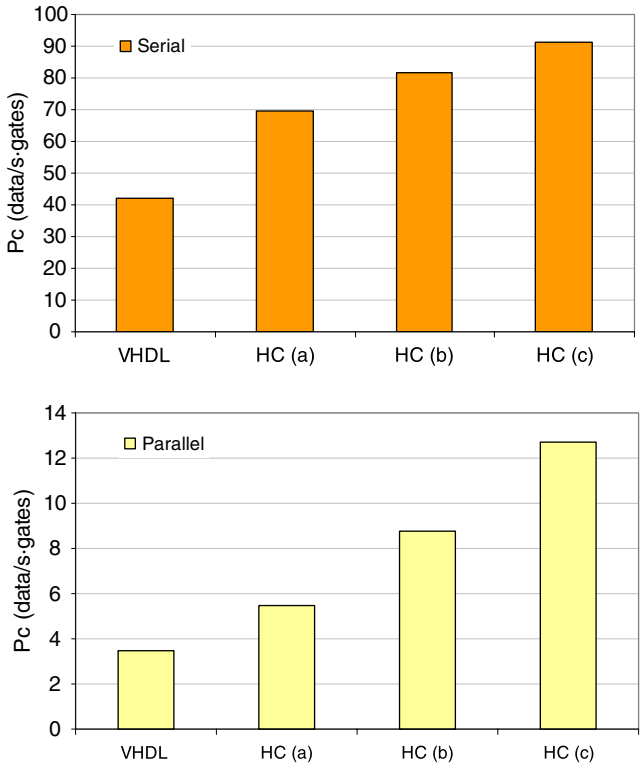

Fig. 6. Performance cost (P_c) of the different evaluated designs.

significantly different: the best design defined at an algorithmic level requires a performance cost approximately 1.6 times higher than the RTL-based version.

The evaluated memory utilization strategies represent different consumptions of global hardware resources as can be seen in Table 2. The most efficient is the option (a) of Handel-C which avoids using EMBs. This is so because the amount of data storage required by each hidden neuron is small (approximately 50% of the capacity of a single EMB) and therefore, they are inefficiently used by our system.

At an algorithmic-like abstraction level, we have easily explored different alternatives for the data storage. We only had to specify which hardware resources are aimed for a concrete data structure. On the other hand, at an RTL abstraction level, this study would require the definition of specific circuits to interface the different types of memory resources. Furthermore, we would also need to define the characteristics of each memory circuit (bit width, number of items of the data structure), whereas at a higher description level, all these features are linked to specific data structures that, once defined, can be compiled using different hardware resources by just indicating it at the instantiation command.

When comparing Tables 1 and 2 (Figs. 5 and 6), where results for a RTL and a higher-level description are shown respectively, it can be seen that the RTL implementation is a more optimal approach for the final system. The serial and parallel RTL designs are approximately about 1.2 times faster than the best higher-level designs. However, one of the main advantages of the high-level description is the reduction of design time for a system. Therefore, for our MLP designs, the design time for the serial case when using high-level description was about 10 times shorter than with the RTL one. In the parallel case, the design time for the high level was relatively small (just enough to introduce the "par" directives), while for the RTL description, a new architecture and control unit had to be designed; which implied a greater difference in the designing time. This is of critical importance when exploring different alternatives for a specific system, as already shown in the several options of data storage described earlier.

We have also carried out a scalability study at a high abstraction level (Fig. 7). In order to obtain similar performance rates, we have used the embedded memory blocks (EMBs) inefficiently by trying to keep the parallel access capability to all hidden neurons. Thus, we can see how this

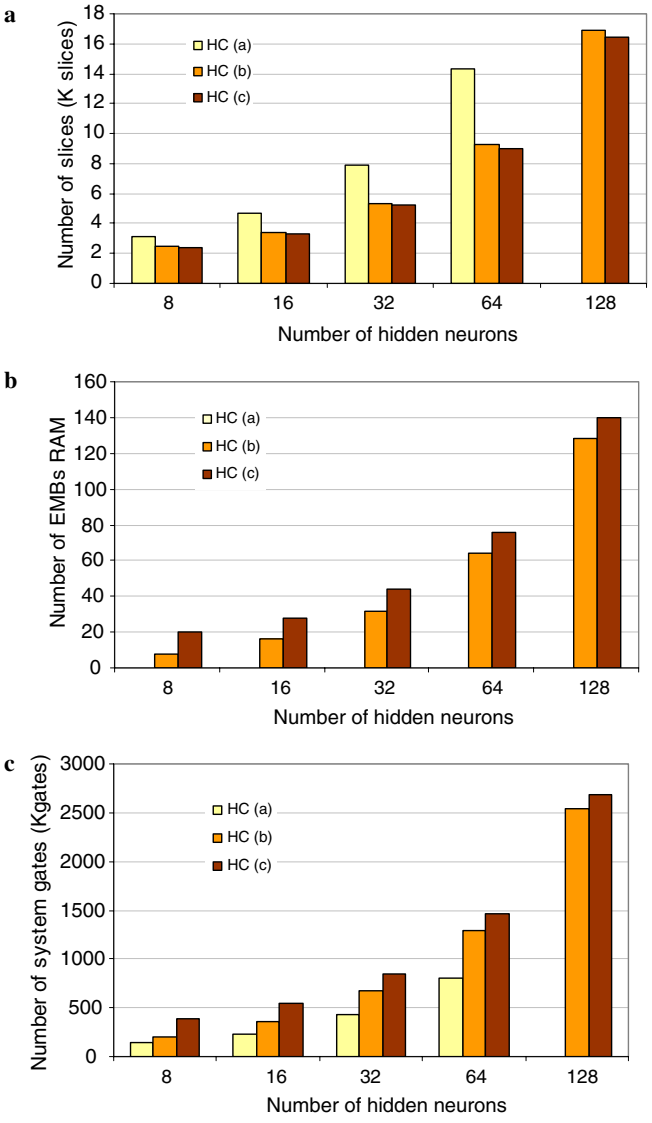

Fig. 7. Scalability of the different approaches: (a) slices; (b) EMBs RAM; (c) system gates.

resource grows linearly with the size of the network (Fig. 7b). If we could afford a lower level of parallelism, this increment would be much less pronounced, since we could fit a higher number of hidden neuron variables into a single EMB. In all cases, the amount of slices also grows linearly with the network size (Fig. 7a), even when using EMBs. This is due to the need for more general purpose logic to properly address the memory elements and the replication of functional units. Finally, we observed that the design that uses only distributed memory also consumes hardware resources that increase linearly with the network size, though the scalability in this case is limited to 64 hidden neurons on the Virtex-E 2000 device. Nevertheless, this would not be the optimal device type to synthesize this design because it does not use EMBs.

5. Discussion

In this paper, we have presented a design strategy for different abstraction levels. A study on different alternatives of data storage and design scalability adopting an algorithmic-like description language (Handel-C) has also been completed. Finally, the paper also showed how a more efficient approach can be designed at a RTL abstraction level. This illustrates how much performance and cost can be optimized by addressing a low-level design in a later refining stage.

FPGA devices have evolved in recent years, not only in the logic elements allocable in a single chip, but also in specific purpose modules (embedded memory blocks, multipliers, and processors) which are already included in last generation devices. This makes it difficult to choose the characteristics of the optimal device towards a good trade-off between performance and cost. In this contribution we compared and evaluated different memory storage strategies. Then, we translated all the different elements into a global characteristic, i.e., equivalent system gates. This allowed us to evaluate how much performance improvement we could gain with parallel computing in a MLP structure. By adopting this approach, we observed that similar performances were obtained with different memory storage strategies. Using only distributed memory seems to be the best option, since the EMBs are inefficiently used to preserve the required parallelism by the MLP. Finally, we also showed the performance improvement obtained by designing at a lower abstraction level (RTL), which, in our approach, is about 1.2 times faster. When estimating how much hardware resources implicate these performance improvements, we confirmed that the option which uses distributed memory resources is the one with less significant "performance cost" (P_c). In fact, the final design done at low abstraction level (using VHDL) saved hardware resources significantly (about 30%) in comparison to the equivalent approach defined at an algorithmic-like abstraction level.

All the different studies were addressed within a real-world application framework thus, obtaining the performance constraints. The neural networks-based speech recognition system requires the parallel MLP implementation of 24 hidden neurons. Although the study about the hardware resources consumption has a significant impact on the final product cost, any of the evaluated approaches would fit the application driven performance specifications. For the speech recognition application, we obtained a correct classification rate of 96.83% with a computation time around 13–16 microseconds per sample, which fulfilled by far the time restrictions imposed by the application. Therefore, the presented implementation can be seen as a low-cost design: the whole system, even the parallel version, would fit into a low-cost FPGA device and could be embedded in a portable speech recognition platform for voice controlled systems.

A pipeline processing scheme, using one neural layer in each stage, would lead to a faster approach. The processing bottleneck is imposed by the maximum neural *fan in* (220 in a hidden node) because it needs 220 multiplications. With a pipeline structure, we can overlap the computation time of the hidden layer with the computation time of the output layer (24 multiplications per node), which would speed up the data path to a maximum of 10%. We did not study the pipeline choice because our design already fulfills the application requirements, i.e., portability, low cost and computation time.

The comparative study between register transfer level using VHDL and a higher abstraction level with Handel-C as description languages sets an illustrative example. The two options led to very different design time vs. efficiency trade-offs. Although the final implementation characteristics depend on the designer's skills, the use of an algorithmic-like abstraction level with Handel-C reduces the design time by a factor of ten. Furthermore, with this modular system in particular, the exploration of different processing strategies (the serial and parallel approaches described in Section 3.1) was straightforward while the VHDL approach required the design of a new control unit.

Acknowledgements

This work has been supported by CICYT TIC2002-00228, CICYT TIC2000-1348, and the EU project SpikeFORCE (IST-2001-35271). We thank the reviewers for their valuable and useful comments.

References

[1] B. Widrow, D. Rumelhart, M. Lehr, Neural networks: applications in industry, business and science, Commun. ACM 37 (3) (1994).
[2] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[3] P. Ienne, T. Cornu, K. Gary, Special-purpose digital hardware for neural networks: an architectural survey, J. VLSI Signal Process. 13 (1) (1996) 5–25.
[4] J. Zhu, P. Sutton, FPGA implementations of neural networks – a survey of a decade of progress, Lect. Notes Compu. Sci. 2778 (2003) 1062–1066.
[5] J. Zhu, G. Milne, B. Gunther, Towards an FPGA based reconfigurable computing environment for neural network implementations,

in: Proceedings of 9th International Conference on Artificial Neural Networks, vol. 2, 1999, pp. 661–667.
[6] R.A. Gonçalves, P.A. Moraes, J.M.P. Cardoso, D.F. Wolf, M.M. Fernandes, R.A.F. Romero, E. Marques, ARCHITECT-R: a system for reconfigurable robots design, in: ACM Symposium on Applied Computing, Melbourne, USA, ACM Press, 2003, pp. 679–683.
[7] D. Hammerstrom, Digital VLSI for neural networks, in: M. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, second ed., MIT Press, 2003.
[8] C. Gao, D. Hammerstrom, S. Zhu, M. Butts, FPGA implementation of very large associative memories, in the book: FPGA Implementations of Neural Networks, in: A.R. Omondi, J.C. Rajapakse (Eds.), Springer Berlin Heidelberg, New York, 2005.
[9] I. Page, Constructing hardware–software systems from a single description, J. VLSI Signal Process. 12 (1) (1996) 87–107.
[10] M. Mylonas, D.J. Holding, K.J. Blow, DES Developed In Handel-C, London Communications Symposium, University College London, 2002.
[11] S.M. Loo, B. Earl Wells, N. Freije, J. Kulick, Handel C for rapid prototyping of VLSI coprocessors for real time systems, in: 34th Southeastern Symposium on System Theory (SSST 2002), IEEE Proceeding, 2002.
[12] L. Rabiner, B.H. Juang, Fundamentals of Speech Recognition, Prentice-Hall, 1993.
[13] R.P. Lippmann, Review of neural networks for speech recognition, Neural Comput. 1 (1) (1989) 1–38.
[14] X.D. Huang, Y. Ariki, M.A. Jack, Hidden Markov Models for Speech Recognition, Edinburgh University Press, 1990.
[15] B. Widrow, M. Lehr, 30 years of adaptive neural networks: Perceptron, Madaline and Backpropagation, Proc. IEEE 78 (9) (1990) 1415–1442.
[16] Mentor Graphics, http://www.mentorg.com/.
[17] Celoxica: Technical Library, http://www.celoxica.com/techlib/.
[18] Xilinx, http://www.xilinx.com/.
[19] Xilinx: Virtex-E Data Sheets, http://www.xilinx.com/support/mysupport.htm#Virtex-E.