Digital Virtual Neuroprocessor: Design experience using Verilog HDL

Jorge Velez, António de Brito Ferrari

***Resumo*— No presente artigo são apresentados o procedimento adoptado e a experiência obtida no projecto e desenvolvimento de um processador neuronal, utilizando Verilog HDL. A descrição das várias etapas de projecto no sentido da descrição final do sistema e do correspondente ambiente de simulação, constitui uma ilustração concreta da aplicação das capacidades de uma metodologia baseada na descrição e simulação de hardware no projecto de sistemas.**
A configuração final do computador neuronal integra o ambiente de simulação de Verilog HDL como ferramenta fundamental para o teste e desenvolvimento de futuro software neuronal.

***Abstract*— This paper presents the design flow and the resulted experience in the development of a neural processor, using Verilog HDL. Describing the design steps toward the final system's description and simulation environment, provides a concrete illustration of the application of Verilog HDL features in the system's design testing, characterization and improvement, at the early stages of the project.**
The end-use neural computer's configuration comprises the Verilog HDL simulation environment as a fundamental tool for future neural software test and development.

I. INTRODUCTION

The implementation of artificial neural networks (ANNs) by software simulators running on serial processing computers, is not suitable for most "on the ground" applications where the real time speed requirements are unreachable even if the fastest serial computers on the market were used. For this domain of applications the use of special parallel hardware implementations of ANNs is required.

This paper describes the design methodology used in the development of one such system. It presents the experience gained in building the Verilog HDL description of a digital neuroprocessor within the system where it is to be integrated. The system description and specification is presented in the following section.

As the complexity of current systems has been growing, the designers have increasingly adopted a high-level description of the system to be designed as the starting point in the development process. Current hardware description languages (HDLs) such as VHDL or Verilog, support the design hierarchy, allowing mixed-level descriptions to be submitted to simulation. They provide a means for designing different parts of the system, using distinct development strategies. The datapath of a processor can be simulated with a behavioral architectural description of its arithmetic and logic unit. Once all the synchronization and clocking schemes have been validated, the arithmetic and logic unit may be substituted by a more detailed structural model. The same may be applied for its constituting parts and so on.

Besides documenting and stating unambiguosly the system's specifications, the use of an HDL allows for their verification without resorting to iterations through the time consuming phase of implementation.

Designing a special purpose processor constitutes a good example of how useful an HDL description and subsequent simulation, can be. Often being distant from the traditional fairly tested architectural solutions, the exploration of the design space through the evaluation of the various alternatives coming up during system development, is of great importance.

The advantages of using an HDL do not end up with the completion of a project. In current integrated circuit (IC) design, the existence of a correct system description is very useful when testing the fabricated chip. The test vectors can be previously evaluated and the HDL simulation responses can then be compared to the real responses from the fabricated IC.

II. BASIC SPECIFICATIONS

The objective is to design and build a computer whose architecture is oriented toward the efficient implementation of ANN models (neural computer or neurocomputer), in particular Multi-Layer Perceptrons (MLPs) with the backpropagation (BP) algorithm [1][2]. The architecture does not make any restrictions to the network configuration, allowing for the mapping of nets with different dimensions and any distribution of neurons among different layers.

Due to the heavy computing requirements of such models in current real-life applications, the neurocomputer to be built is based on a parallel configuration. Connected as an attached processor to a host computer, it is constituted by a number of processing units communicating via a common data bus (fig.1).

The proposed architecture envisages simplicity and low cost adopting solutions to the typical bottlenecks. The characteristics of neural nets (large number of neurons and interconnections, simple internal processing) make

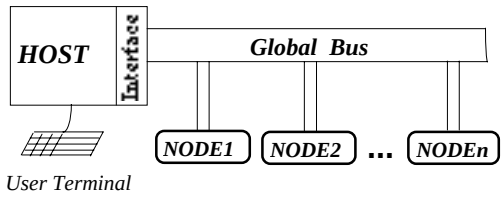

Fig.1

the algorithms communication-intensive. Hence the limitations in communication bandwidth tend to be the main bottleneck in multiprocessor neural architectures. The fundamental system characteristics (fig. 2) can be summarized as follows:

- *Neuroprocessor* - each node in fig.1 includes a special-purpose floating-point processor implemented as a VLSI chip, and its own local memory.
- *Parallel architecture easy to expand* - by implementing inter-node communications through a tagged inter-communication bus type, the architecture's typical bandwidth bottleneck, can be minimized.
  The system is scalable simply by adding new VLSI chips onto the global bus.
- *Broadcast communication type* - similar to the classical single bus architectures, its design principles come from the observation that nowadays memory is cheap in comparison to arithmetic resources [3][4]. Providing each node with sufficient local storage, traffic on the bus can be reduced to the new neuron activations [1]. The learning algorithm must be adapted to accomodate global communication.
- *Reconfigurability and Generality* - in programming mode, apart from loading the neuroprocessor microprogram, each node also receives the information about the network topology. Such topology is totally re-configurable through the host computer.
  Although the architecture was thought for improved performance with the BP algorithm, it may also be used for executing other algorithms on other ANN models.
- *Computation and Communication Balance* - a node is designed to fully support the overlap of communication with internal processing. Communication and processing are asynchronous, exchanging data through input/output first-in-first-out (FIFO) buffers.

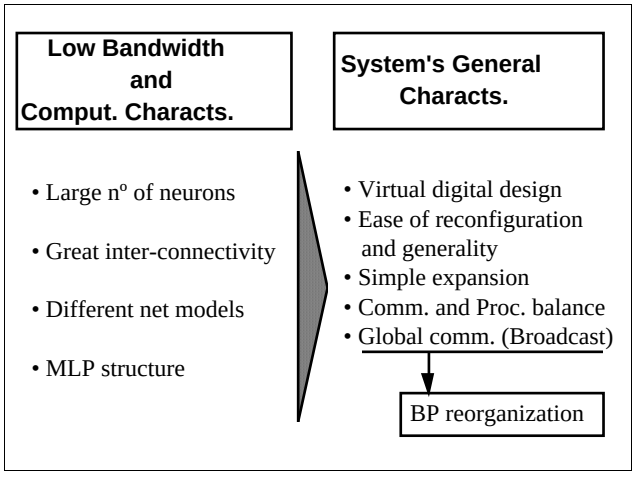

Fig. 2

The operation is controlled by the host and encompasses three main phases (modes):

1. **Programming** of the system by the host, where the network topology is defined, and the initial values for the weights are loaded into each node. After programming the operation may proceed to the training or recall modes.

2. In **training** mode the host presents the training patterns and searches for the output neuron activations (*forward* step). When one of such activations is read, the host calculates the error and sends it to the bus. Once the backpropagation of errors is complete (*backward* step), another input pattern is presented. At the end of the training set, all the nodes update their weights (*update*).

3. **Recall**, where the operation is just as in the forward phase of the learning process.

III. NODE ARCHITECTURE

Since the data word length has a major impact on operation speed and silicon area, a reduced floating-point representation was investigated [2]. The conclusion was that a 16-bit floating-point format (1-bit sign + 5-bit exponent + 10-bit mantissa) provided the required precision and dynamic range. Each processor consists of two separate units working asynchronously. A communication unit, the node's interface with the global bus, and a processing unit whose arithmetic resources are multiplexed for each neuron in a node (fig. 3).

A. Communication Unit

The chosen bus protocol is the 3-wire Daisy Chain (GRANT, REQUEST, BUS-BUSY), arbiter centralized at the host bus interface. How this choice affects performance was subject to experimentation through the final HDL simulation [2] and it will be discussed later.

The communication unit consists basically of input and output FIFOs (data and tag fields) and of a PLA-based controller which also implements the Daisy Chain

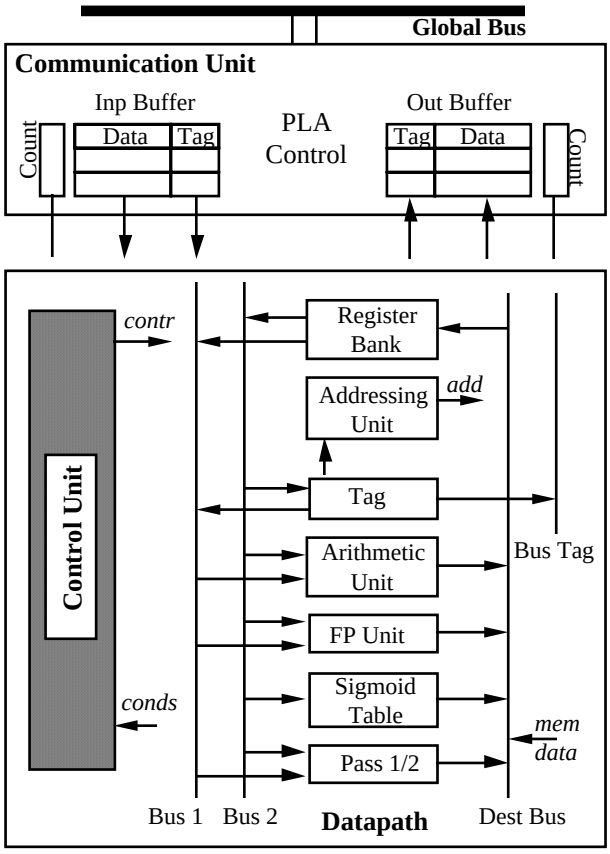

Fig. 3

protocol. The length of these buffers has been investigated [2].

B. Processing Unit

The processing unit, divided in control unit and datapath, is a (micro)programmable processor with a dedicated instruction set. A hardwired design would compromise the desired neuroprocessor flexibility.

This unit's fundamental design decisions derive from the system specifications and the BP processing characteristics:

- *Control Unit* - this unit is microprogrammable and based on SRAM where the machine code implementing the neural algorithm is to be loaded. The access time of the control memory is dependent on its dimension. Hence it is important, both for chip area and for processing speed, to keep the SRAM small.

- *Addressing Unit* - according to the tag coming through the bus, the neuron being processed and the type of data involved (weight, update or activation), a memory position has to be determined. This is a frequent operation, therefore the addressing scheme is of crucial importance to the processor's performance.

  Two addressing modes are supported. An absolute mode with no arithmetic required and a displacement mode involving the summation of the incoming tag, the neuron base address and their dimensions.

- *Floating-Point Unit* - the neurons intrinsic calculations (summation and multiplication) are executed by a 16-bit floating-point unit (FPU). The short mantissa greatly contributes to the feasibility of such FPU.

- *Sigmoid Table Look-up* - by using the exponent bits and the sign value, a 64-position SRAM implementing the non-linearity, is directly accessed. Another implementation choice would involve heavy calculations.

Other important relevant parts are:

- *Integer Arithmetic Unit* - the integer arithmetic unit is meant, basically, for pointer and index calculations needed to implement loops in the algorithms. Since the addressing unit also uses a 16-bit adder, a hardware multiplexing scheme was implemented so that some silicon area can be saved.

- *Register Bank* - there are 8 general purpose registers. Although this number represents a good tradeoff, a few more would allow for a slight decrease in memory traffic.

IV. VERILOG AND VERILOG-XL

Verilog (**Veri**fy **Log**ic) is the HDL most used in industry, particularly in the US. Introduced in 1983 by Gateway Design Automation it was later acquired (1989) by Cadence Design Systems that offered it to standardization (1991). As a result a number of third-party simulators, most of them running on PCs, is now available.

The early availability of an efficient simulator supporting all the language constructs and well supported by the system environment (Verilog-XL), together with its similarity to the C programming language, were instrumental to its widespread acceptance. Its availability to universities under EUROCHIP, and its easy integration with Cadence Design Framework, together with the characteristics mentioned, led to the decision to choose Verilog as the HDL to be used.

The logic simulation process with Verilog-XL, involves three separate steps: creation of a system model, providing stimulus to exercise the model and indicate the format in which results are to be viewed. The simulation sessions may be interactive allowing the user to monitor the system variables, forcing new stimulus or changing the previous ones. The simulation may then proceed normally, step by step with or without trace [5].

Input/Output of data from and to files under the operating system, is possible. Such features allow the gathering of statistics, register final values, etc, and the

reading of data values to memories, PLA contents, etc. The output display includes normal printing in the working window, fixed position textual printing, bars or waveforms. The last three are updated as time evolves. Past events can be viewed moving a cursor in a time bar. The output facilities and the great simplicity of the related commands, are undoubtedly a further strong motivation to use Verilog-XL.

The language basic concept is the **module**. It represents a piece of hardware connected with other modules through inputs and outputs, called **ports**. Such modules can be part of a hierarchy as a block of hardware that can be used one or more times in another module.

The modules functioning can be described either behaviorally or structurally. Behavioral descriptions can be done at the register transfer, algorithmic or architectural levels. The structural, netlist type, may be described using primitives such as gates or through transistor models (switch level).

All levels, behavioral or structural, can be mixed in a same description and submitted together to simulation. Thus high level behavioral descriptions of parts being designed, can be simulated together with the structural description of already implemented blocks.

The time concept constitutes the main particularity of HDLs. Since every piece of hardware works in parallel through time, Verilog code is executed in an event-driven way. Every action is triggered by specific user-defined events or through the value of a global variable representing the system simulation time.

The package language+simulator could be used as a standalone product or integrated in Cadence Design Framework. Before going to the implementation phase, for practical reasons, the standalone use is advised.

V. NODE DESCRIPTION

As referred, the first stage toward the neuroprocessor implementation was the C language neural network simulator. By defining the processor's data format, the basic specifications were finally achieved.

Design and description are indivisible. Naturally, ideas come first to light through scratches on paper. However, by creating models of the pieces of hardware, whatever the description level is, such ideas can easily be judged, for invalidation or for adoption. The advocated architecture is the basis for this kind of development. In the present design, by architecture one refers not only to the system but also to the node's architecture.

Level independent, the description structural detail is also a matter of concern. Using behavioral constructs one can build a module either modeling its function with behavioral procedural blocks and register transfers, or by detailing its constitution. By other words, one may use only the behavioral level to model the block function or, in a structural way, implement it with smaller and simple behavioral level modules. This latter type of description can be easily and automatically transformed into an ordinary schematic by mapping those simple modules, to the correspondent standard or user-developed cells, of a specific technology.

The system design flow will be exposed as the sequence of its main subparts description and their linking procedure (fig.4). All the description top hierarchical modules and the ones resulting from joining them, were simulated before being linked again. Some requirements and consequences of such procedure can also be viewed in fig.4.

A. Communication Unit

In the communication unit design, the first step was to implement the bus protocol. Thus, an arbiter and a controller for the 3-wire Daisy Chain, were built. The controller is part of the communication unit while the arbiter belongs to the host interface. An extra line (inhibition line) is included to prevent the arbiter to grant the bus if any of the input buffers gets full.

Having validated the bus protocol scheme description, the remaining bus communication unit functions were then described. Since its control will be a PLA-based state machine, it wasn't important to go deep in detail. At this stage, spending time describing the PLA contents and surrounding logic, was unnecessary. Instead, there were written some behavioral code lines to implement all the block functioning.

For an initial communication unit test, **statistical distributions** were used for representing the items processing time and the time interval between production of two consecutive activations. Besides the **waveforms** display, **graphical output bars** monitor the buffers occupation through time and the maximum number ever, in them. **Statistics** such as mean inter-arrival or longest wait times, are also generated.

Apart from validating the basic architectural concepts, several important conclusions could already be obtained. It was shown that the fixed priority established by the Daisy Chain protocol, had no effect unless the input buffers got full. However, such situation only affects performance if the output buffers are also filled up, which rarely occurs if the buffer lengths are well dimensioned. Some description related details and possible deadlock situations were identified. Communication aspects since they depend on algorithms, topologies and relative processing times between nodes, were not studied at this stage.

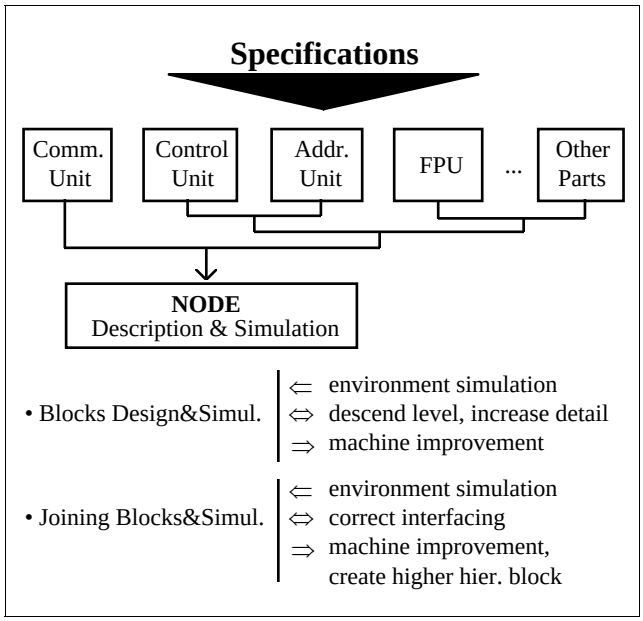

Fig.4

B. Control Unit

Independently from the bus communication module, the processing control state machine was specified. The main purpose of this first version was to validate its functioning principles and related clocking strategy. By supplying the host lines and emulating the conditions coming from the datapath, not yet built, all the unit synchronization was defined.

The description detail was such that behavioral models were only applied to pieces of hardware as simple as multiplexers, latches and registers. In some constituting blocks the detail evolved from less detailed descriptions. Behavioral code with no detail, was used first to determine or prove the effectiveness of the functional specifications of those blocks. This illustrates a development process starting from a high abstract level and ending with a schematic equivalent structural description. Thus, it will be quite fast to go from such a description to implementation.

At this stage it was important to question the feasibility of having SRAM for the control memory on-chip, though the details for the µword format were not known yet. In order to have an idea about the physical aspect of such memory, ES2's parameterized SRAM cells were used. The respective access protocol and timings, were followed.

C. Addressing Unit

A first module version was created so that the addressing scheme under the adopted clock strategy could immediately be validated. Separate parts descriptions constitutes not only a way to immediately verify the idealized architecture but also a safe way to develop the top hierarchical modules, as they grow up by integrating smaller ones.

A second version was finally built which combines the arithmetic unit and the addressing unit in the same module, and implements the 16-bit adder multiplexing scheme.

D. Datapath Parts

Having in mind the advocated generality, though it will be directed to the BP algorithm, a first version of the instruction set was defined through translation of pseudo-code algorithms into assembler-like code. The required datapath's transfers and operations determine the hardware to build and to optimize.

The basic datapath parts are: the FPU, the integer arithmetic unit, a bank of general purpose registers, the addressing unit, the function table and a special register — TAG register. The corresponding modules were independently developed though there are some common building blocks.

After specifying the modules synchronization and interfacing, the µword format could finally be defined. Since this processor is user-microprogrammed, the possible µword's bits combinations give rise to a huge instruction set dimension. The allowed instruction set, assuring machine integrity, will be a small subpart of all the combinations.

The simulation of all the parts working together was quite extensive since the µword lines had to be emulated. Incorrect synchronization of the applied stimulus allowed some defects to pass, being detected later.

The final description detail is similar to the control unit one. It requires a small step to implementation. For instance, the FPU was fully designed. Its basic building blocks are well known pieces of hardware (ex.):

```
module arith_unit(out,ovw,undw,op1,op2,sub/add,enable);
    ...
    exponent_comp       exp_comp(diff,exp1,exp2);
    sign_detector s_det(out,signA,carry_add,diff);
    normalizer          ...
    ...                 ...
endmodule

module sign_detector(out,signA,carry_add,diff_signs);
    ...
    assign #delay out=(diff_signs &&
                              ~carry_add)?~signA:signA;
    //Implemented with simple combinatorial logic.
endmodule
```

The FPU project is certainly an excellent example to illustrate the importance of using an HDL and simulation.

By including high level constructs to perform the data format conversions, the result could be displayed in both machine and neuroprocessor representations. The conversion between floating-point formats was done building specific Verilog code functions used just as in C. Their existence facilitates enormously the full FPU test and, in case of error, ease for determining its location.

E. Joining Control and Addressing+Arithmetic Units

Joining these parts, was the next phase. Well defined the cycles and synchronization of the respective instructions, by filling the correspondent µword bits, the whole scheme was then tested. Imperfections mainly relative to the interfacing, could now be fully detected and corrected. A working processor able to perform addressing and arithmetic operations, and branch according to the resulting conditions, was obtained. The external SRAM interfacing was also included.

F. Joining the remaining Datapath Parts

The complete synchronization scheme was defined and verified, at this stage. As a new piece of hardware was included, the new datapath transfers involved were immediately simulated and tested. It wasn't as hard as testing the units separately because the stimulus were now filled as the respective µword fields.

The occurring errors were normally and easily detected, either by consulting the register values permanently displayed on screen or by messages originated in modules. Such fault detection messages were quite useful. When a hardware error was detected, such as a floating line on which the next µword address depends, then a warning message was sent or an interrupt was generated.

To simulate "all" the specified instruction set, it was only necessary to emulate the communication unit asynchronous signals and the respective buffers.

G. Joining Communication Unit

Finally, the communication module was joined to the processing unit description, creating a complete neural node. By making instances of several neural node modules connected to the bus and its arbiter, the system simulation would then be done.

The external SRAM requirements had to be evaluated. Through a few calculations involving the number of neurons and connections foreseen, and being aware about constraints such as the limit number for the nodes on the bus, the capacity and speed of current commercial SRAMs were found to satisfy the needs (64K words, <20ns T_{acc}).

Designing with the target technology permanently in mind, is fundamental. The impact of some of the crucial system subparts on the final design, was studied whenever it was possible. Such study was normally supported by the ES2 design tool kit, as in cases of the multiplier, function table and PLAs.

VI. SYSTEM SIMULATION

The simulation of the complete system required more than the description of its parts. A **system module** integrating the various neural processor modules, the external SRAMs and the global bus arbiter, was created.

It is necessary to provide a practical way for adapting the Verilog system description according to the topology and configuration in use. The system module uses the Verilog *include* directive to include the configuration dependent parts. Those parts, such as node instances, output commands, node port variables, programming, etc, are automatically generated through **C code**. The C routines also fill the external SRAM with the global configuration data and random weights, if specified.

The function table is created by generating in C, the sampled function values, and then, converting them into the specified 16-bit floating-point format. If the function and the number of points to represent it remain unchanged, this procedure has to be executed only once.

Since programming directly in microcode would be extremely difficult and time consuming, an **assembler** was built which implements a subset of the allowed instructions. If a special instruction is required or an optimization through merging consecutive µwords, is desired (e.g. within cycles), it can always be done using one of the assembler directives which specifies explicitly in hexadecimal format, the contents of a control memory word. Further capabilities include the use of labels and comments.

The test module, representing the host computer actions, resulted quite complex. The host tasks had to be emulated in Verilog. Such module implements the mode transitions, drives the items into the global bus, snoops the bus for activations, controls the supervision line (e.g., to command the update operation), collects statistics and measurements of various system parameters, monitors the system variables for on screen visualization, etc. In addition to the described complexity, almost all the tasks above require re-writing the code whenever there is a configuration modification or a change in some other system parameters.

The devised functional description for the complete system, can be illustrated as in fig.5. Configuration and application data are situated at a different entry level since the algorithms shall be provided as part of a set of neural software and other tools.

Through distinct interface routines, the host front-end which comprises the assembler and all the configuration routines, controls both the hardware implementation and the HDL description. The HDL simulation is useful not only while the system is being developed but also during the future practical applications. It will be used for new algorithms validation and characterization.

Accomplished the HDL description, several performance measurements can finally be made toward performance prediction and architecture evaluation. Communication and processing balance are dependent on algorithms, number of cycles to consume an item, number of cycles to produce an item, network configuration and network neurons correspondence to nodes, number of nodes, buffers length and bus protocol. A deep investigation on such matters demands extensive variation of system parameters and network configuration which may result in a huge simulation time. Additional data such as instruction use measurement for code optimization (size, efficiency) applied to the algorithm crucial parts, can be obtained.

Some predefined tracing analysis were provided: **bus utilization** vs time, **node processing** vs time and **buffer status** vs time. In such conditions, one can verify how different is the bus utilization during the various operating phases and its dependency with other parameter settings. The buffers length dependency on the network parameters, network partition and number of nodes, was appreciated.

The load distribution and its dependency on the simulation conditions, can be analyzed. For instance, different partitions of the same network can be tried toward a more even load balance. The number of output requests with the output buffer full and the waiting time for an item with the input buffer empty, may be counted. It measures the processing units forced latency.

Through the measurements in various simulation conditions, some architecture's parameters, such as its granularity and load balancing, can be evaluated. The HDL description and simulation can be used for full architecture performance characterization.

The following examples illustrate the importance and type of information possible through system simulation. Figure 6 is an example of bus utilization and nodes processing monitorization versus time, during the backward step of the BP algorithm (network 2x7x2 — 2 input neurons, 7 hidden and 2 outputs). The buffers size was 5 which allowed the output buffers to remain not completely full

The execution time constitutes a direct measure of system performance. It allows algorithm codifications appreciation, to qualify the network partitions, to optimize the system parameters (such as number of nodes and buffers size) for a particular application, etc. An example of performance versus number of nodes is presented in figure 7, for two distinct networks during the forward step or recall functioning mode.

The speed-up non-linearities are due to inevitable distinct number of neurons implemented by the different nodes in each simulation.

In relation to the buffers dimension analysis (figure 8 shows the direct graphical output in a specific simulation moment) several conclusions were obtained.

The maximum input buffer occupation is in fact imposed during the forward step for networks with hidden dimension higher than the output dimension. For the backward step the situation reverses.

The output buffer occupation is minimum as long as the input buffers are well dimensioned. Otherwise the maximum number of items possible is determined by the number of neurons implemented by the respective node. Therefore, a programmed solution for a possible deadlock when both buffers get full, is justified. Since such situation will be rare, a hardware solution would be too costly.

By locating the microprocessor control state machine eventual bottlenecks and by investigating their foreseen operation delays (e.g., consulting the ES2 tool kit), besides conducting the improvement efforts toward the most influent hardware aspects, it gives an idea about the future processing unit clock frequency. Among the cycle time constraining parts that may exist, it can be included the external SRAM access time, the ALU/ADDR unit operations or the FPU. Parts such as the function table, were eliminated as eventual bottlenecks.

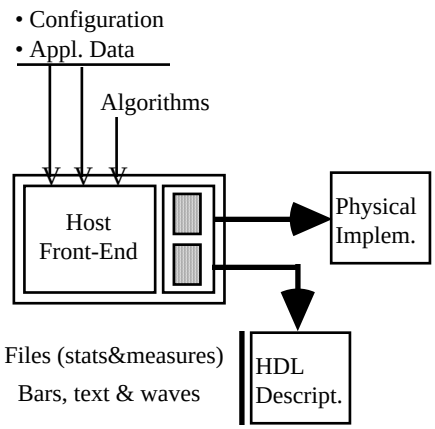

Fig.5

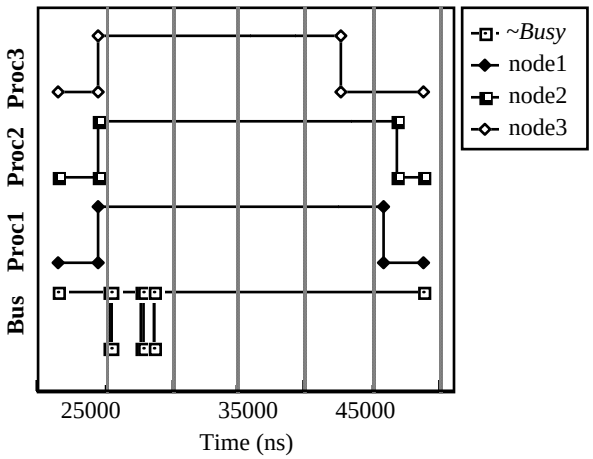

Fig.6

To run the simulation only one terminal command is necessary. It indicates the system parts location (generally in separate code files), and what type of simulation delays is going to be used (**min**, **typ** or **max**).

VII. SIMULATION DIFFICULTIES

The main drawbacks of such system simulation, arise not from the testing and validating design phase but from building the whole system description so that its performance and architectural solutions, could be evaluated. Thus, the algorithms had to be machine programmable right away, the configuration (neural network and number of nodes) had to be automatic, programming issues and host control had to be emulated, etc. The creation of the surrounding environment (mainly host jobs) in conjunction with the various successive system parts test, constituted the less interesting and most time consuming phases.

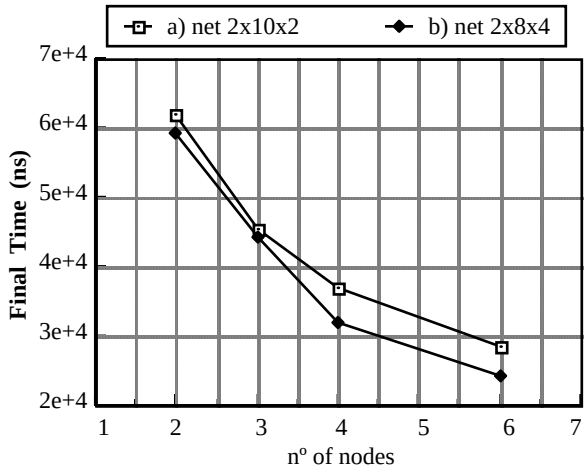

Fig.7

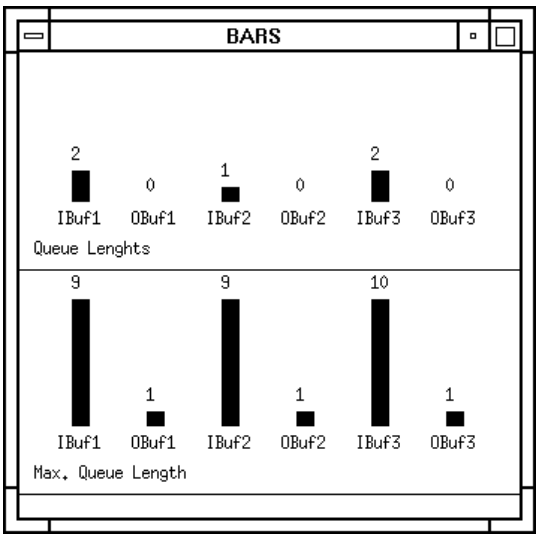

Fig. 8

The assembler had to be created to allow faster, lexical and syntactic error free code writing, floating-point format conversions were programmed, instances and variables in Verilog code had to be automatically created through specific C routines, the function table was also generated automatically, etc. A small modification, easy to accomplish on the HDL description, may have a much bigger impact on the configuration dependent routines. The code for collecting data, such as the statistics, had also to be automatically generated. However, Programming Language Interface (PLI) could certainly be used, in this case.

Verilog doesn't allow multidimensional constructs to be applied to variables and instances, otherwise, the configuration process could be practically automatic. Although such efforts should be unnecessary at this stage, one must notice that many of the configuration work is useful for the future host front-end package.

The simulation time is too long for learning procedures limiting the simulations to few epochs (learning iterations over the entire training set). It certainly is not at all surprising, since even with the C language simulator, the algorithm could be, for large networks, very time consuming. For accelerating the simulation, the graphical output may be disabled.

VIII. CONCLUSIONS

Several HDL general advantages were proved and illustrated throughout this system's development. The description and simulation, easy to accomplish, are fundamental for a methodic and coherent design flow. They constitute a "helping rope" linking the former development stage, to state the specifications, to the final test design step.

Through the use of Verilog HDL, by clearly specifying the different modules and their interconnections, division for independent parts design, can easily be done. Therefore, it represents an adequate development tool for a design group job division, the way successful commercial products are currently developed.

The initially proposed architecture, could be validated in its principal aspects. The built HDL description constitutes a powerful framework for measuring, testing and, consequently, improving the system. In particular, several fundamental design decisions and optimization, mainly related to communication aspects, this system typical bottleneck, were supported by the built simulation framework.

The possibility for creating the surrounding environment and emulating the real stimulus to the processors, gives more credibility to the simulated tests. Also, being aware of the target technology, there is no separation from realizability even when high level models are used.

The choice of level and description detail shall always consider, besides the modelling purposes, minimization of time to produce the code. For an immediate great detail description the schematics method would be preferable.

To pass to implementation will be an easier task. The designer can now concentrate his efforts onto the implementation aspects rather then worrying about the architectural solutions to be devised.

In parallel with the physical system, the HDL description will be useful for future use. Apart from end applications use, it is proper for investigation and teaching purposes. In addition, it will be fundamental for validating future user programmed software, before running it in the nodes.

Verilog HDL reveals serious handicaps to deal with varying configuration models. In the future, it should feature multi-dimensional constructs for variables and module instances. Mainly due to configuration dependent code, a great deal of C code is necessary for achieving the proposed objectives.

The user-friendly simulator graphical output, very attractive, constitutes an extra motivation for Verilog use.

REFERENCES

[1] António de Brito Ferrari, Y.H.NG, "A Parallel Architecture for Neural Networks", in *Proc. Parallel Computing*, 1991.

[2] J. Velez, "Processador Neuronal Virtual Digital para Implementação VLSI", Tese de Mestrado em Engª Electrónica e Telecomunicações, DETUA, Universidade de Aveiro, 1995.

[3] R.Kuczewski, M.Myers, W.Crawford, "Neurocomputers Workstations and Processors: Approaches and Applications", IEEE 1st ICNN, 1987, San Diego, CA.

[4] Robert Hecht-Nielsen, "Neurocomputing", Addison Wesley, 1990.

[5] "Verilog-XL Reference Manual", 2 Volumes, Version 1.6 March 1991, Version 1.6a November 1991 Release Notes, from Cadence Verilog Manuals.