Chapter 6

VHDL Implementation of the PLPN-Based Bidirectional Associative Memory

6-1 VHSICHDL Hardware Description Language

6-1-1 Definition

VHDL [1] is the acronym for VHSIC HDL standing for Very High Speed Integrated Circuits Hardware Description Language. This language was proposed to model digital systems at many levels of abstraction, ranging from the algorithmic level to the gate level. The complexity of a digital system being modelled could vary from that of a simple gate to a complete digital electronic system, or anything in between. The digital system can also be described hierarchically including explicitly its timing in the same description. The VHDL language can be regarded as an integrated amalgamation of the following languages:

- sequential language
- concurrent language
- netlist language
- timing specifications
- waveform generation language

Therefore, the language enables the designer to express the concurrent or sequential behaviour of a digital system with or without timing. It also allows the designer to model the system as an interconnection of components. Test waveforms can also be generated using the same constructs. All the above constructs may be combined to provide a comprehensive description of the system in a single model description.

The language defines not only the syntax but also defines very clear simulation semantics for each language constructs. Therefore, models written in this language can be verified using a VHDL simulator. It is a strongly typed language and is often verbose

to write. It inherits many of its features, especially the concurrent language part, from the ADA programming language. Because VHDL provides an extensive range of modelling capabilities, it is often difficult to understand. Fortunately, it is possible to quickly assimilate a core subset of the language that is both easy and simple to understand without learning the structure of the most complex chips.

6-1-2 VHDL Standardisation History

The requirements for the language were first generated in 1981 under the V_{HSIC} program. In this program, several of the US companies were involved in designing V_{HSIC} chips for the Department of Defence (DoD). At that time, most of the companies were using different hardware description languages to describe and develop their integrated circuits. Consequently, different vendors could not effectively exchange designs with one another. In addition, different vendors provided DoD with descriptions of their chips in different hardware description languages. Reprocurement and reuse was also a big issue. Thus, a need for a standardised hardware description language for the design, documentation, and verification of the digital systems was generated.

A team of three companies, IBM, Texas Instruments, and Intermetrics, were first awarded the contract by the DoD to develop a version of the language in 1983. Version 7.2 of VHDL was developed and released to the public in 1985. After the release of version 7.2, there was an increasing need to make the language an industry-wide standard. Consequently, the language was transferred to the IEEE for the standardisation in 1986. After a substantial enhancement to the language, made by a team of industry, university, and DoD representatives, the language was standardised by the IEEE in December 1987; this version of the language was known as the IEEE Std 1076-1987.

The official language description appears in the IEEE standard VHDL Language Reference Manual, available from the IEEE. The language has also been recognised as an American National Standards Institute (ANSI) standard. According to IEEE rules, IEEE standard has to be reballoted every five years so that it may remain a standard. Consequently, the language was upgraded with new features, the syntax of many constructs was made better uniform, and many ambiguities present in the 1987 version of the language were resolved. This new version of the language is known as the IEEE Std 1076-1993.

The department of Defence, since September 1988, requires all its digital Application-Specific Integrated Circuit (ASIC) suppliers to deliver VHDL descriptions of the ASICs and their sub-components, at both the behavioural and structural levels. Test benches that are used to validate the ASIC chip at all levels in its hierarchy must also be delivered in VHDL. This set of government requirements is described in Military Standard 454. Since 1987, there has been a great need for a standard package to aid in model interoperability. This was because different CAE (computer-aided engineering) vendors supported different packages on their systems, causing a major model interoperability problem. Some of the logic values used were 46-value logic, 7-value logic, 4-value logic, and so on. A committee was set up to standardise such a package. The outcome of this committee was the development of a 9-value logic package. This package, called STD_LOGIC_1164, was then the balloted and approved to become an IEEE standard, labelled IEEE Std1164-1993.

6-1-3 VHDL Capabilities

The followings are the major capabilities that the language provides along with features

that differentiates it from other hardware description languages:

- The language can be used as an exchange medium between chip vendors and CAD tool users. Different chip vendors can provide VHDL descriptions of their components to system designers. CAD tool users can use it to capture the behaviour of the design at a high level of abstraction, for functional simulation.
- The language can also be used as a communication medium between different CAD and CAE tools. For example, a schematic capture program may be used to generate a VHDL description for the design, which can be used as input to a simulation program.
- The language supports hierarchy; that is, a digital system can be modelled as a set of interconnected components; each component, in turn, can be modelled as a set of interconnected sub-components.
- The language supports flexible design methodologies: top-down, bottom-up, or mixed.
- The language is not technology-specific, but is capable of supporting technology-specific features. It can also support various hardware technologies; for example, you may define new logic types and new components; you may also specify technology-specific attributes. By being technology-independent, the same model can be synthesised into different vendor libraries.
- It supports both synchronous and asynchronous timing models.
- Various digital modelling techniques, such as finite-state-machine descriptions algorithmic descriptions, and Boolean equations, can be modelled using the language.

- The language is publicly available, human-readable, and, above all, it is not proprietary.
- It is an IEEE and ANSI standard; therefore, models described using this language are portable. The US government also has a strong interest in maintaining this as a standard, so that reprocurement and second sourcing may become easier.
- The language supports three basic different styles: structural, dataflow, and behavioural. A design may also be expressed in any combination of these descriptive styles.
- It supports a wide range of abstraction levels ranging from abstract behavioural descriptions to very precise gate-level descriptions. However, it does not support modelling at or below the transistor level. It allows a design to be captured at the mixed level using a single coherent language.
- Arbitrarily large designs can be modelled using the language, and there are no limitations imposed by the language on the size of a design.
- The language has elements that make large-scale design modelling easier; for example, components, functions, procedures, and packages.
- Test benches can be written using the same language to test other VHDL models.
- Nominal propagation delays, min-max delays, setup and hold timing constraints, and spike detection can all be described very naturally in this language.
- The use of generics and attributes in the models facilitate back-annotation of static information such as timing or placement information.
- Generics and attributes are also useful in describing parameterised designs.

- A model can not only describe the functionality of a design, but can also contain information about the design itself in terms of user-defined attributes, such as total area- and speed.
- A common language can be used to describe library components from different vendors. Tools that understand VHDL models will have no difficulty in reading models from a variety of vendors since the language is a standard.
- Models written in this language can be verified by simulation since precise simulation semantics are defined for each language construct.
- Behavioural models that conform a certain synthesis description style are capable of being synthesised to gate-level description.
- The capability of defining new data types provides the power to describe and simulate a new design technique at a very high level of abstraction, without any concern about the implementation details.

6-1-4 Hardware Abstraction

VHDL is used to describe a model for a digital hardware device. This model specifies the external view of the device and one or more internal views. The internal view of the device specifies the functionality or structure, while the external view specifies the interface if the device through which it communicates with the other models in its environment. Figure (1) shows the hardware device and the corresponding software model.

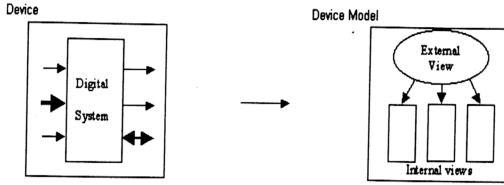

Figure (1). The hardware device and the corresponding software model

The device-to-device model mapping is strictly one-to-many model. That is, a hardware device may have many device models. For example, a device modelled at a high level of abstraction may not have a clock as one of its input, since the clock may not have been used in terms of, say, integer values, instead of logical values. In VHDL, each device model is treated as a distinct representation of a unique device called an entity. Figure (2) shows the VHDL view of a hardware device that has multiple device models, with each device model representing one entity. Although, entity 1 through *N* (see Figure 2) represents *N* different entities from the VHDL point of view, in reality they represent the same hardware device.

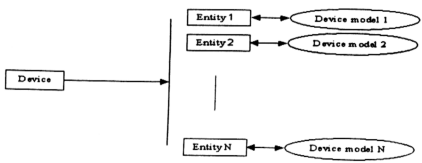

Figure (2). VHDL view of a hardware device that has multiple devices models.

6-2 VHDL Implementation of the Parallel Learning-Processing Neuron

6-2-1 Introduction

Before going further in the implementation of the PLPN based BAM using the VHDL description, it is worthy to think first about the different aspects and structures of this programming style and its useful capabilities. However, the structure of the precedent implementation will be described by the VHDL language and in the way (as needed) the reader will be introduced to the VHDL programming characteristics. Besides, the important aspect of the different programming styles used for the implementation, mainly the functionally partitioning model that was independently suggested for the chip modelling strategy by Armstrong [2], will also be presented. The precedent technique dividing the synapses into excitatory and inhibitory are in this implementation generated internally after presenting the association. This technique is used in order to minimise the cost of the circuit although other version of this technique has been implemented using one synapse that was observed to be hardware intensive.

6-2-2 Network Source Code

For the implementation of the present circuit, the VHDL description has been divided into two different parts:

1) The neuron unit: and its VHDL description is presented in Appendix B.

2) The Control unit: and its VHDL description is presented in Appendix C.

In the control unit part, a circuit that will synchronise between the neurons was created. This unit will protect the network from any confusion in its outputs or any overlap between the different states of the single neuron similar to the lights and cars in the traffic ways.

The neuron itself has to do different operations by being in different *phases* to implement the suggested Parallel Learning Processing strategy. These operational phases are:

1) Initialisation Phase

2) Memories-number Storing Phase

3) Operating Phase.

In the initialisation phase, each neuron's output will be initialised ('1' or '0') which with the other neurons outputs will constitute the initial configuration of the neural network. During the second phase, the neuron stores the number of memories that will be sent periodically to it during the operating phase.

In the operating phase, the neuron performs the inner product multiplication between the input vector signal, which contains the output of the connected neurons to the local one, and the corresponding synaptic strengths.

Before going further in the definition of the internal structure of the neuron, first the interface ports of the neuron circuit, through which it communicates with the control unit and the network neurons, is to be defined. This is performed by the entity construct.

6-2-2-1 The Entity

The input and output ports which are defined in the entity unit of the program are shown in the diagram of the Figure (3).

In this unit, the generic statement has been used. This kind of statements is used to define some internal constant parameters of any circuit such as delay timing, bus width and other VHDL specific objects such as the filename and so on.

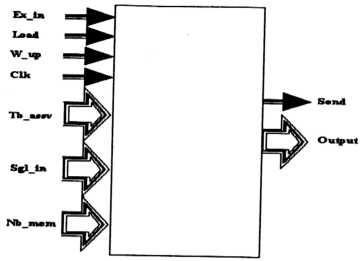

Figure (3). Representation of the interface ports signals of a single PLP Neuron. The buses are presented by big arrows and the single signals are presented by small ones

In the present design the generic statement has been used to declare constants that will represent the width of the configuration bus signal declared as sgl_in, the accumulator, the memory storing-number bus nb_mem, and intermediate variables.

6-2-2-2 The Architecture

The architecture construct of the system defines the internal structure of the neuron. One of the most important characteristics of this VHDL construct is that it can be written in different ways. Theses methods are generally dependent on the modelling techniques used by the designer, which are the basic styles of VHDL description of the architectures. These styles could be summarised in the following:

1) The behavioural Style.
2) The Data-flow Style.
3) The structural Style.

In the behavioural style, the behaviour of the entity is expressed using sequentially executed, procedural code, which is very similar in syntax and semantics to that of a high-level programming language like C or Pascal. The process body construct, which is one of the most useful VHDL description tools, is the primary mechanism used to model the behaviour of an entity. This modelling style was adopted in the implementation of the PLP neuron because of its flexibility and simplicity to design complex circuits.

The Data-flow style specifies the functionality of the entity without explicitly specifying its structure. This functionality shows the flow of information through the entity and it is expressed primarily using concurrent signal assignment statements beside other specific statements such as the block statement that is quite similar to the process statement. In this modelling style, in contrast with the behavioural modelling, the functionality of the entity is expressed using statements that are executed in concurrent (parallel) fashion.

In the structural modelling style, the entity is modelled as a set of components connected by signals like a netlist. The behaviour of the entity is not explicitly apparent from its model. The components are entities of circuits, where their architecture is already defined, and is used as components when the need to use them from other circuits descriptions arises. The component instantiation statement is the primordial VHDL construct used for describing such a model of an entity.

For the implementation of the PLP neuron a modelling technique based on the communication between the different process, which constitute our architecture has been suggested. Each process in the architecture has a special task to perform in parallel or sequential fashion depending on the task to be realised. In the diagram bellow, a process model graph called the functionally partitioning model has been presented to clarify the

communication among the different processes. This modelling technique has been suggested previously and independently by Armstrong [2].

6-2-3 Functionally Partitioning Model Graph

For the clarification of the graph in Figure (4), let first define some useful symbols in order to understand the different steps of the processing. The narrow vector represents a signal that belongs to the sensitivity list of the indicated process. Another vector that represents a signal belonging to the sensitivity list of the concerned process is the double interpenetrated vector, which represent only the bus signal of the associated bit to the associated vector. Finally, the side-shaded arrow represents the clock signal which is a sensitivity signal of the principal process while the last represents the signal (bus or not) used inside the process itself. The process is represented in the above diagram by a circle. The process statement is however, one of the most important statements used in VHDL behavioural programming. It contains sequential statements that describe the functionality of a portion of an entity in sequential terms. A set of signals to which the process is sensitive is defined by the sensitivity list so that, each time an event occurs in the sensitivity list, the statements within the process are sequentially executed. The process then suspends after executing the last statement and wait for another event to occur on a signal in the sensitivity list.

The Calcul process is the main process in our implementation, which is reflected by its central position of the above graph. It has the clock signal clk in its sensitivity list, therefore it is executed each time the clock get an event '0' to '1' or '1' to '0'. The positive edge triggering of the clock was chosen in order to meet the current designing convention.

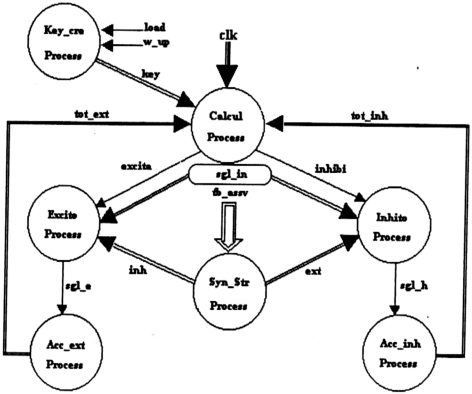

Figure (4). Process Model Graph: Functionally Partitioned Model of the Neuron architecture.

This process is also governed in its functioning by the value of the bus signal key which let the process do different functions. The key itself is composed of two input signals, the signal w_up and the signal load. When the bus signal key is equal to "00" the Calcul process initiate, after the positive triggering of the clock, the default output of the neuron. Therefore, the neuron is in the initialisation phase. When the signal key is equal to "10" the neuron is in the memories-number storing. Then, the Calcul process begins the operating phase when the signal key changes to "01". In this step, the neuron enters the operating phase and 'awakes' two processes excito and inhito by sending their sensitivity signals excita and inhibi respectively. The two processes will perform in

parallel the inner product between the bus signal sgl_in and the signal ext, and the sgl_in and the signal inh, respectively. The signal ext represents the corresponding excitatory synaptic strengths of the associated vectors from X layer and Y layer and the signal inh represents the inhibitory one. They are generated from the Syn_Str process, which is sensitive to the associated vectors, sent continuously by a synchronised memory. Synchronised memory represents the corresponding memory defined in the synaptic strength subcircuit of the previous PCB implementation. After the calculation of the corresponding inner products, the two processes stores the product into two separate internal buses, and will 'awake' at the same time another two processes Acc_ext and Acc_inh. This is done by sending two signals sgl_e and sgl_h respectively. The two respective active processes will accumulate respectively the results of the previous processes to their precedent signal values tot_ext and tot_inh. The process Calcul continues processing this step and verifying that the number of sent memories has achieved or not the limit stored during the memories-number storing phase. When the limit is achieved the Calcul process will compare between the signal tot_ext and signal tot_inh in order to get the correct output that will be assigned to the neuron.

6-2-4 The Control Unit

In the above paragraphs, the implementation of the PLP neuron, that is the basic computational unit in the network has been described. However, to realise the complete network, a control unit is needed. The control unit that will harmonise between the different implemented PLP neurons is defined in the top-level circuit VHDL description. This unit has been implemented to avoid confusion and mis-functioning among the neurons in the two layers X and Y. It controls and regularises between the phases of the

neurons in both layers. It means that this control unit will set up the control signals load and w_up of all the neurons in the layers X and Y in following way. Firstly, both layers will be initialised and let store the number of processing memories simultaneously. After that, one layer will start the processing phase (its bus signal key getting "01") while the other is idle. In the present implementation the idle phase is replaced by the memories-number storing phase in order to make the PLPN based BAM more flexible and dynamically reconfigurable. It means that it can be used to handle different number of associated memories with different kind of memories without the need of relearning it by increasing or deleting the stored ones. Here exactly comes the usefulness of our suggested strategy of PLP based BAM, i.e., that the neural network is able to learn or store and process simuitaneously at any moment any information. Beside the control unit in the top-level, the appropriate connections between all the neurons in both layers have been created. Therefore, in order to define these connections, the use of another important statement especially when implementing artificial neural networks called package is needed. The package is implemented separately from the top-level source code but it will be called later by this source to use its implemented signals, in the package body, when specifying the work library in the top-level source code description.

6-2-5 Serial and Parallel Processing Versions

The central processing area in the implementation of the artificial neuron is the multiplication of the inner product of its input vector bits with their corresponding synaptic strengths. After realising this multiplication, a summation operation has to be performed to sum the products of this multiplication. The multiplication is realised using AND gates, whereas the bits summation imply two methods of implementations. The

first, is the parallel version, where each multiplication resulting bit is attached to an adder in a serial way with the other bits so that the summation will propagate from an added to an adder. This operation is carried out at every clock rising during the operating phase (see Figures (5), (6), (7) and (8)).

The second method is the sequential version, which implies the use of the accumulator, where at each step the output of the accumulator will be updated by the new added product bit. However, after some edge rising of the clock signal (equal to the number of the summed bits), the accumulator will output the result of the total summation of the inner products. It can be concluded, comparing between the two versions of the inner product summation, that more the required operation to be implemented is aimed to be fast, more the hardware expense increases and vice versa.

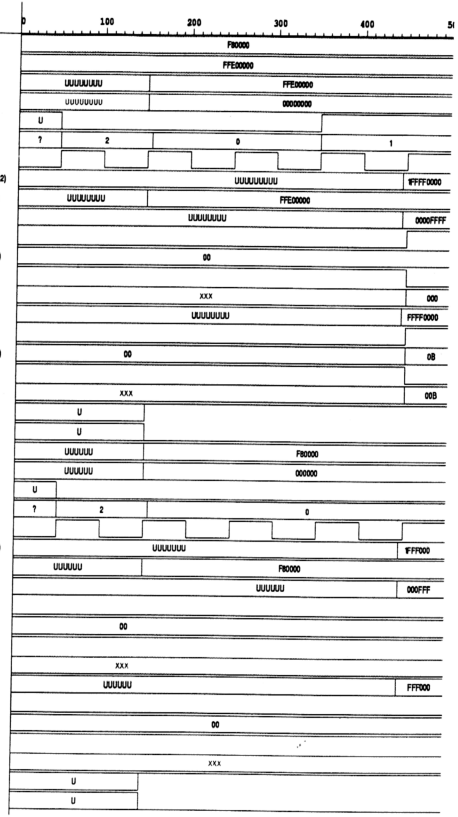

Figure (5). VHDL Simulation Diagram of the Parallel version of the Digital PLP-based BAM circuit in the time interval [0 ns, 500 ns].

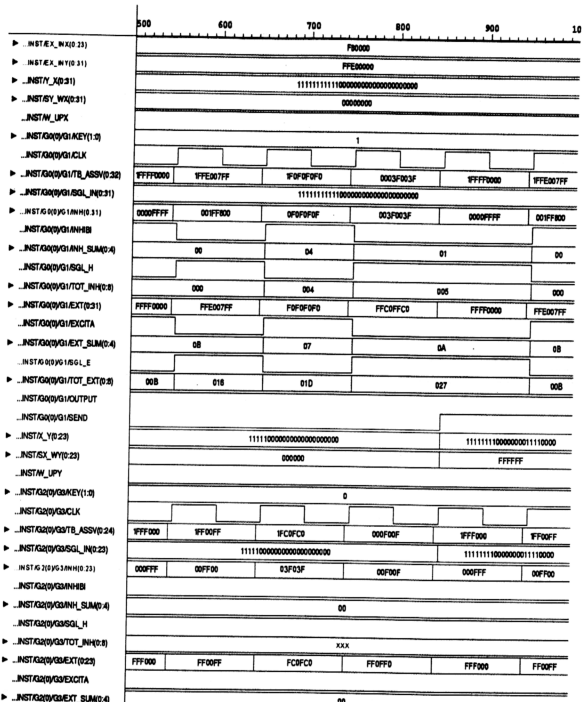

Figure (6). VHDL Simulation Diagram of the Parallel version of the Digital PLP-based BAM circuit in the time interval [500 ns, 1000 ns].

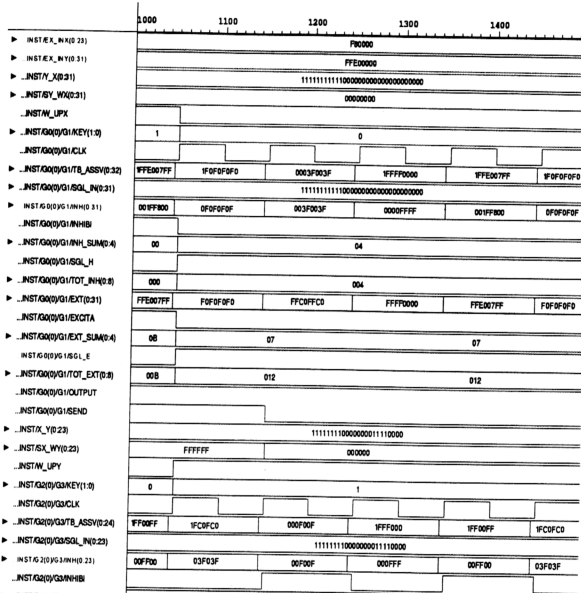

Figure (7). VHDL Simulation Diagram of the Parallel version of the Digital PLP-based BAM circuit in the time interval [1000 ns, 1500 ns].

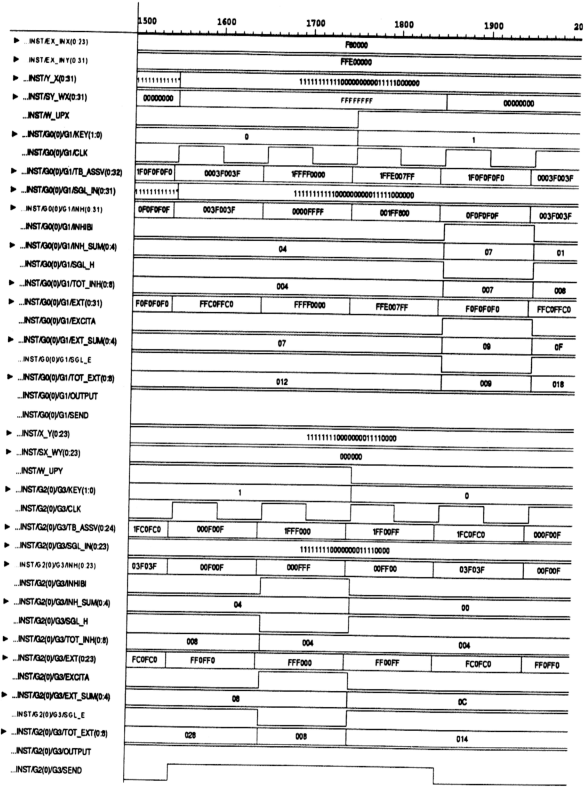

Figure (8). VHDL Simulation Diagram of the Parallel version of the Digital PLP-based BAM circuit in the time interval [1500 ns, 2000 ns].