Chapter 7

The Implementation Expandability & Interfacing Techniques

Two major parameters (see Chapter 3) which characterise the limitation and capabilities of neural network implementations are the expandability of the adopted architecture and its performance to realise a required task. Therefore, new architecture called *systolic-like architecture* is proposed to realise the expandability of the PLP based BAM network. Secondly, and for the accomplishment of the second parameter an interfacing technique between associative memory and the user is suggested. This technique *called encoding-comparing technique* (ECT) is proposed to enable the user to evaluate the associative neural network stable states, whether they are true memories or spurious states.

7-1 Network Implementation Enhancement Using Expandable Systolic-like Architecture

7-1-1 Introduction

The present part concerns the BAM implementation expandability based on the parallel learning-processing neuron treated in the precedent chapter. Our purpose here is to suggest an architecture that can guarantee the flow of the memorised associated-vectors to BAM units so that their processing will look like a large[7] BAM implementation formed by a serial concatenation of these units. For this purpose, architecture partially inheriting the data flow of the systolic arrays called *systolic-like architecture* [1] is proposed.

7-1-2 Expandability of the BAM Units

Two crucial factors mentioned in the previous chapter about the digital VLSI implementation of ANN, are the scale and the scalability of the implementation. Scalability reflects the ability of the implemented network to be expanded (as unitary

[7] In terms of the number of neurons in each layer

element) to support any number of neurons and therefore to be more adaptable to the real need of neural computation. This factor was taken into consideration in our implementation by exploiting the power of the systolic array architecture. Although the idea was inspired from the systolic array architecture, the suggested architecture is not totally similar. Figure (1) shows the proposed architecture mapping PLPN based BAM.

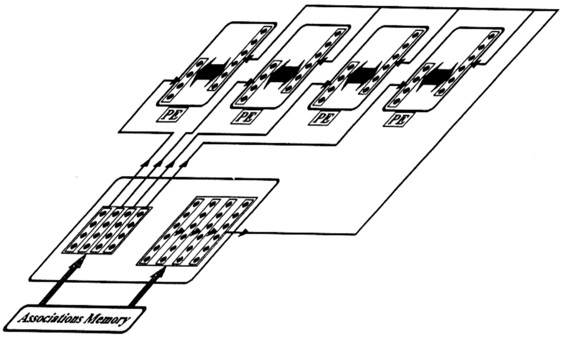

Figure (1). The expandable architecture of the PLPN based BAM.

The VHDL description of the expanded network along with its constituent parts including the hierarchical top circuit is presented in Appendix D. This top circuit of the expanded BAM architecture is constituted of two major components. The associated memories server and a collection of processing elements (PE) which are just the basic PLPN based BAM units. The associated-memories server itself (see Figure 1) is composed of two kinds of digital memories, a parallel shift register and a serial shift register. The parallel shift register holds one of the two associated memories divided into equivalent segments. These segments are sent by the memory in parallel to the processor units, in such way that each segment will be sent to one processor element. On the other hand, the serial shift register is composed from the same number of segments as the parallel shift register and holds the other associated memory whereas it sends serially the segments to all processor units. In this processing, each of the segments of the parallel shift register is presented to one layer of the corresponding processing element whereas the other layer will receive continuously one by one the different segments of the shift register. This operation is performed in such way that each processing element processes the association between a given segment of the simple memory and all of the segments of the serial shift register as shown in Figure (2). After that all of the segments of the serial shift register were sent another new association of memory is to be loaded in parallel into both of the memory units; the parallel and the serial shift registers. The above processing continues operating similarly pursuing the above processing until all the associated memories are learnt. Then, one layer of all PEs segments (the processing layer) updates its outputs and the operating phase of the two layers is to be reversed. Therefore, the layer which was in the processing (accumulation) operating state will turn to the idle (receiving the number of the associations) state and vice-versa.

Before closing this section, first the systolic array sight of the present implementation must be mentioned. The implementation scheme shown in the Figure (1) was inspired from the pattern-matching systolic array suggested by Kung [2]. For this architecture, there exist two versions: the semi-systolic and the pure systolic array. Figure (3) and Figure (4) depict the difference between them.

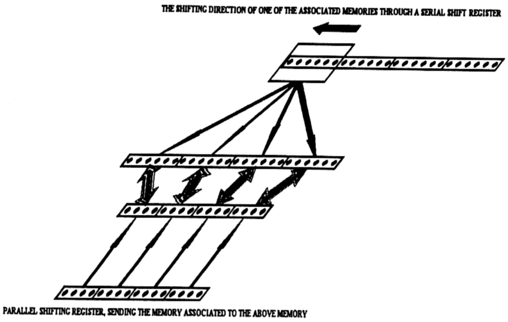
THE SHIFTING DIRECTION OF ONE OF THE ASSOCIATED MEMORIES THROUGH A SERIAL SHIFT REGISTER
PARALLEL SHIFTING REGISTER, SENDING THE MEMORY ASSOCIATED TO THE ABOVE MEMORY

Figure (2). Four PLPN-based BAM Units Implemented by the Systolic-like Architecture.

However, without going further on the details of the pattern matching problem and its resolution by the above systolic architecture, it is worthy to refer to the interesting book of M. D. Blasi [3].

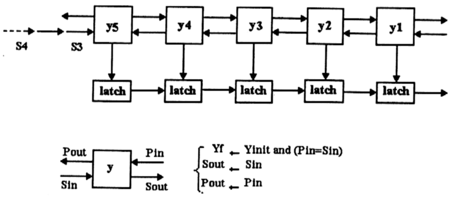

Figure (3). Semi-systolic array for pattern matching problem:
a) Configuration of the systolic array, b) Base cell and transfer function

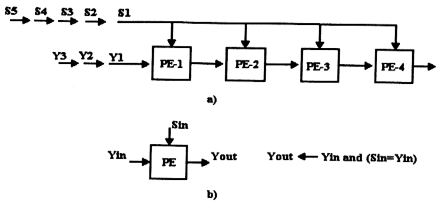

Figure (4). The Sample Text Characters Systolic Movement (Pattern-Matching Problem)

The main fact that relates between our suggested architecture and the above architecture is the distribution of the information flow to the processing elements. This is realised so that all the processing elements will realise the memory learning recalling in parallel fashion. Furthermore, the architecture of the data flow, especially the semi-systolic architecture in which the *S* data (see Figure (3)) flows like one of the associated vectors (*X* layer) flows to the BAM units although the two others differ. However, this difference is made because of the interconnectivity among the processing elements in the semi-systolic array is unused in our implementation. The reason is the processing independence of the different BAM units except that each unit has to perform the association between each segment of the same system memory. Therefore, this modified version of the systolic array architecture is suitable for the implementation of the strategy of parallel learning processing to realise the expandability of the network.

7-1-3 VHDL Implementation of the Systolic-Like Architecture

7-1-3-1 Intermediate Association Memory

The major unit in this implementation is the association memory in which a pair of associated memories are stored in segments and sent continuously to the BAMs units.

However, this memory unit is built based on two parts: the serial shifting part and the parallel shifting part. From Figure (1), the parallel memory presents each segment of an associated image to each segment of a layer in each BAM unit. However, the other part sends the associated image segments serially segment by segment, to the next layer of each BAM unit. Besides, both registers perform under control signals three tasks namely shifting, loading and freezing. For the last case, the registers are frozen as they receive a given signal from the corresponding port and will perform no operation. In Figure (5) and Figure (6) respectively, the basic unit of the memory register and a register memory composed of eight bits (units), are presented.

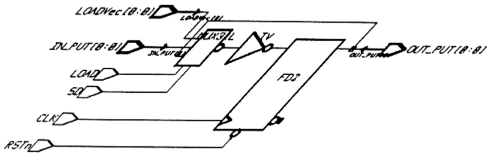

Figure (5). The Memory Basic Unit Schematic.

7-1-3-2 Associations Memory Unit

This memory unit is designed to hold the system associations and it is based on the same basic memory register as the previous. This memory is a conglomerate of memory associations of four BAM units, each of which is composed of 24 neurons by 32 neurons. Exploiting the scalability of the systolic-like architecture, a BAM system composed of 4 x (24 by 32) neurons (i.e., (96 by 128) neurons) has been designed. Therefore, the associations-memory unit stores the system associations composed of 96 bits (neurons) in one layer and 128 bits (neurons) on the other layer.

7-1-3-3 System Control Unit

Although, there is no separate control unit in the system source code, a control process statement in the system VHDL description is implemented. This control process governs the different parts of the system circuit to maintain their consistency and avoid any processing interference. Although this control unit has not been independently designed, an alternative to realise that is possible and does not require complication.

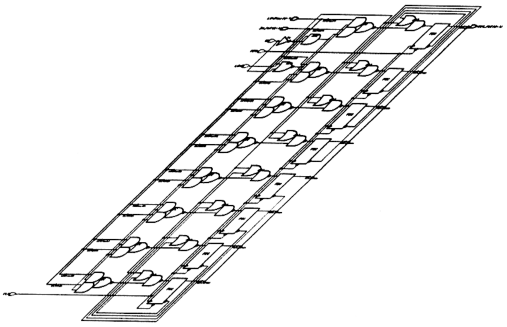

Figure (6). Memory Register Composed of eight Basic Units.

7-1-4 Conclusion

New architecture inheriting the data-flow parallelism of the parallel-array processors and performing the PLP strategy based BAM expandability, is proposed. It is thought that the PLP strategy can benefit largely from these kinds of parallel architectures, mainly systolic arrays, for mapping other neural networks, because of their common parallelism of the dataflow. The mapping of the PLP strategy to implement other neural networks such as multi-layer perceptron and its backpropagation learning rule is possible but we think it might consume much more time. This time consuming of the architecture is due to the complexity of backpropagation learning rule, compared with the simple Hebbian rule. Consequently, the PLP strategy is more dedicated for Hebbian-based architectures.

7-2 Network Implementatio n Enhancement Using Spurious State Detection

7-2-1 Introduction

A technique called the Encoding-Comparing Technique (ECT) is proposed to detect the spurious states of a neural associative memory from those corresponding to stored states. The cyclic encoding strategy has been adopted because of its simplicity and its hardware low cost.

The purpose of the ECT technique [4] injection is the differentiation between a stored memory and a spurious one, which are stable states after the relaxation of the network (BAM). Similarly but in another context Fontanary and Koberle [5] introduced another technique based on modifying slightly the Hebbian-learning rule.

The ECT technique is dedicated however to create an interface between the user and the associative neural memory rather than enhancing its recalling process. The candidate memory is encoded before being sent to the neural memory. The generated code is concatenated with its vector generator (candidate memory) to form a coded memory, which will be sent to the network for learning and storing. After the learning phase (or the learning-processing phase) and when the network gets its stable state the encoder is used again to encode the corresponding part to the vector generator in order to get the corresponding code. Then a comparison between the generated code and the code digits of the stable state determines whether it corresponds to a stored state or no. In similar work, L. Personnaz et al., [6] have proposed the use of an encoder inside a neural associative memory (Hopfield network). The difference between their technique and the present one, is the use of the resultant stable state and its inverse to determine whether it corresponds to a stored memory or no.

7-2-2 Error-Correcting Codin g Theory

The digital coded systems for either data transmission or data storage has much in common. Since the channel or storage medium is subject of various types of noise, distortion, and interference, the channel or the storage medium differs from its input because they are both sensitive to the errors that can result from impaired transmission. The theory and application of error-correcting coding [7] are dedicated for protection of digital information against the errors that occur during data storage or transmission. Many ingenious error-correction techniques based on vigorous mathematical theory have been developed and have many important and frequent applications. The current problem with any high-speed data communication system is how to control the errors that occur during data transmission through a noisy channel. In order to achieve reliable communications, good codes and efficient decoding algorithms are needed. The knowledge of error-correction coding is in great demand and becomes an important asset for many practising engineers and computer scientists who are involved in the design of large digital systems.

The block diagram in Figure (7) illustrates the basic elements for transmission or storage of digital information through a coded system. For this system, all information transferred between blocks must be in digital form if error control is to be used. The data which enter the communication (storage) system from the information source, are first processed by a source encoder that is designed to convert the source information into a digital coded form *d* called the information sequence. The channel encoder transforms the information sequence into a binary-coded sequence *c* called a code word. The binary digits in a code word from the channel encoder are fed into a modulator (or writing unit) that transforms each unit bit into an elementary signal waveform. The waveform that entered the channel (storage medium) will be corrupted by noise. The waveform channel consists of all the hardware and physical media that the waveform passes through in going from the output of the modulator (writing unit) to the input of the demodulator (or reading unit). Typical examples of waveform channels are telephone lines, microwave links, high-frequency links...etc.

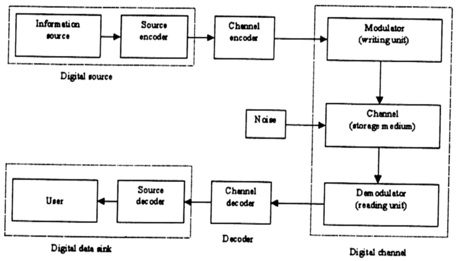

Figure (7). Schematic representing of the encoding-decoding technique mapping.

Each of these examples is subject to various types of noise disturbances. Next, the resulting received signal is processed first by the demodulator and then by the channel decoder. The demodulator (reading unit) makes a decision for each received signal of duration T seconds to determine which of the two possible digits, one or zero, was transmitted. The output of the demodulator is called the received word *r*. This word *r* may not match the transmitted code word *c* because of transmission error. The Channel decoder transforms the received sequence *r* into a binary sequence *d'*, or the estimated information sequence. Since the noise may cause some decoding errors, the channel decoder must be implemented to minimise the probability of the decoding error. The channel decoder uses the syndrome of a received code word *r* to correct the errors in the received word and then produces an estimate of the information sequence. If all errors are corrected, the estimated information sequence *d'* matches the original source information *d*. The source decoder transforms the estimated sequence *d'* into an estimate of the source output and delivers this estimate to the user. Thus the source decoder performs the inverse operation of the source encoder and delivers its output to the data sink.

A cyclic coding is a technique that generates cyclically a code word **c** of *n* digits for a data word *d* of *n-k* digits. In its systematic structure, the code word contains in its first *k* digits what is called the parity-check bits representing the code core. In its second *n-k* bits, it stores the data word bits. The code is generated in a polynomial form by the so-called code generator represented by *g* (*x*) that is of degree *n-k*, so the cyclic code is represented by *g* (*x*), *n*, and *k* or simply referred to by (n,k). The polynomial form means that the sequences of the data to be coded are to be the coefficients of non-decreasing degree polynomial; e.g., the data word $(1\,0\,1\,1)$ is to be represented by $1*x^0+0*x^1+1*x^2+1*x^3=1+x^2+x^3$. To generate a code word in a non-systematic structure, the data word is multiplied by the generator polynomial. Consider the (7, 4) cyclic code whose generator polynomial of degree 3 is $g(x)=1+x+x^3$, and $d=$ $(1\,0\,1\,1)$

so the corresponding information polynomial is then $d(x)=1+x^2$ so the code polynomial becomes $c(x)=d(x)*g(x)=(1+x^2)*(1+x+x^3)=1+x+x^2+x^5$ and the code word will be c = $(1\,0\,0\,1\,1\,1\,0)$. To generate the code word in a systematic structure, one has to divide (look at the referred book for more details about the division) $x^{n-k}d(x)$ by $g(x)$ thus one has $x^{n-k}*d(x)=q(x)*g(x)+l(x)$ where $l(x)$ represents the remainder polynomial and its coefficient are the parity-check bits. As an example, consider the above example with a word data d = $(1\,1\,0\,0)$ so $d(x)=1+x$. Since $n-k=3$, then $x^3*d(x)=x^3+x^4$ and by dividing it over the code generator yields the remainder polynomial $l(x)=1+x^2$ and the code polynomial $c(x)=1+x^2+x^3+x^4$ or, c = $(1\,0\,1\,1\,1\,0\,0)$.

7-2-3 The Encoder Implementation and its BAM Injection

We have seen that the encoding of *k*-bit information sequence involves computing the parity-check bits. Thus, the encoding is accomplished by using a division circuit, which is $(n-k)$-stage shift register with feedback connections specified by the generator polynomial that is represented by $g(x)$. In systematic encoding, which is generally used for high-rate codes, the information bits are transmitted without alteration as shown in Figure (8), where $g_1, g_2, \cdots, g_{n-k-1}$ are the coefficients of the code generator. The rectangles represent one-bit shift register and the Gate is a switcher that is closed during information sequence flow and open at its end. The adopted code generator is $g(x)=x^5+x^2+1$, which is a primitive polynomial over the Gaulois field GF(2) [7]. It generates a code of 5 bits to cover the capacity of the implemented BAM. However, the cyclic encoder which is able to generate $2^5 = 32$ different codes, represents the minimum cyclic encoder to be used for the EC technique implementation. The circuit schematic implementation is shown in the diagram of Figure (9).

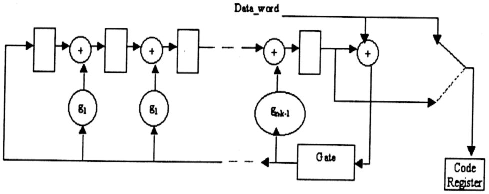

Figure (8): Representation of the implementation of the cyclic encoder

The Gate have been designed by including a synchronous counter so that the Gate's output will be disconnected from its input whenever the counter indicates the number of bits of the word data

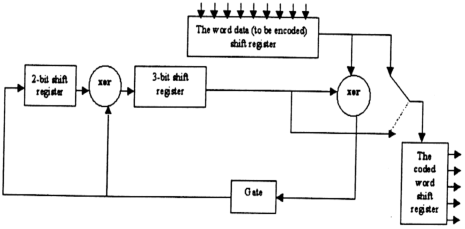
The word data (to be encoded) shift register
2-bit shift register
xor
3-bit shift register
xor
Gate
The coded word shift register

Figure (9). General description of the implemented cyclic encoder

The above general schematic of the cyclic encoder circuit is developed by the VHDL description presented in Appendix E along with its injection in the BAM network. The schematic in Figure (10) is the Synopsis tool generation of the cyclic encoder based on its VHDL description.

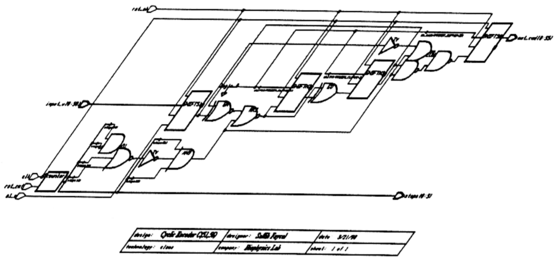

Figure (10). VHDL schematic description of the implemented cyclic encoder

When the BAM network configuration achieves stability, the encoder is enabled and the resultant data will be divided so that only data part will be stored into the word data shift register. Its coded version will be stored into the corresponding register. Whenever the counter (inside the Gate) indicates the number of bits of the code word (5 + number of bits of the word data), the code part (parity-check bits) of the coded word will be sent to the comparator. The later compares the parity-check part with the same part (the first 5 bits) of the original configuration. When the comparison indicates a positive response, an output signal gets a logic value '1' indicating that the configuration corresponds to one of the stored memories. In the other case, the circuit will reverse the input configuration, and compute again the parity-check bits before realising the comparison.

When the comparison gives a positive response, this means that the network has fallen into a reverse state of one of the stored memories. The process is clarified in the diagram of the Figure (11). It is well known, for neural networks based on the Hebbian-learning rule, that the reverse of any stored configuration is also a stable state of the network. Consequently, it can converge to any memory as well as its reverse configuration.

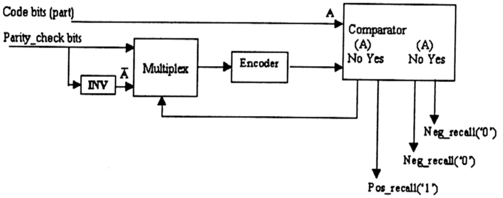

Figure (11). The injection strategy of the encoder into the BAM circuit

VHDL simulation of the signals and buses dynamic of the implemented PLPN-based BAM including the injected ECT circuit are shown in Figures (12) and (13).

7-2-4 Encoding Comparing Technique Simulation Results

The encoding comparing technique (ECT) does not give always the right answer and therefore, it has a margin of failure. The technique error rate depends on the average hamming distance between the stimulus and the stored memories. The average hamming distance is the minimum hamming distance with a given image or its reverse. Let assume *X* layer, the layer having the minimum number of neurons, and *Y* the other layer. The simulation results depicted by the Figures (14) to (21) are obtained using a C-programming where the hamming distance is concatenated within the indices of the axes. The first remark is the independence of the ECT success rate on the hamming distance between the input stimulus and the *Y* stored memories. This means that the ECT success rates depend only on the layer of low number of neurons. Secondly, when the input average hamming distance with any of the *X* stored memories is low, the ECT failure rate is low. However, when this distance grows high the failure rate increases and achieves its maximum when the hamming distance is more or less half of the number of the *X* layer neurons. Finally, as the number of stored memories grows, the contrast of the failure rate of the ECT technique or the success rate of the recalling does not change although their values go higher and lower respectively. Nevertheless, the failure rate of the ECT technique does not exceed much 25% (27 %) when the number of the associated memories reach the network capacity limit. In the present case, the network capacity is equal to the number of neurons of the *X* layer (24). Consequently, the ECT technique seems a good tool to 'interface' the user with the ANN chip especially Hebbian-based networks. In fact, this 'interface' is not perfect especially when the key vector (input configuration stimulus) is far from any of the stored memories or the stored memories are not orthogonal (as the present case is). As the Hamming distance increases, the ECT technique loses increasingly the user 'confidence' because of its increasing failure rate. In the practical cases, the key (stimulus) is not so far from any of the stored memories and therefore, the ECT technique should be useful. Finally, it is worth mentioning the strong dependence between the perfect recall rate and the orthogonality of the stored memories [8].

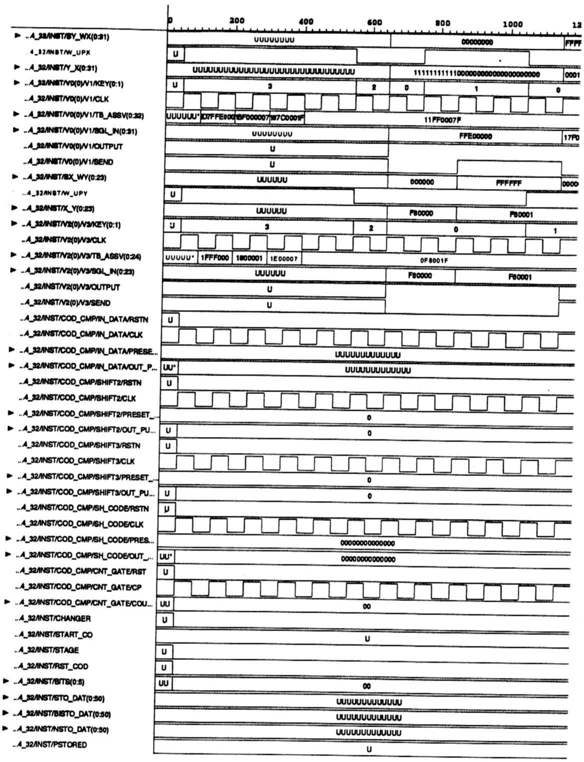

Figure (12). VHDL simulation Diagram of the Cyclic Encoder C (51,56) injected in BAM circuit (24 Neurons/ 32 Neurons) in the time interval [0 ns, 1200 ns].

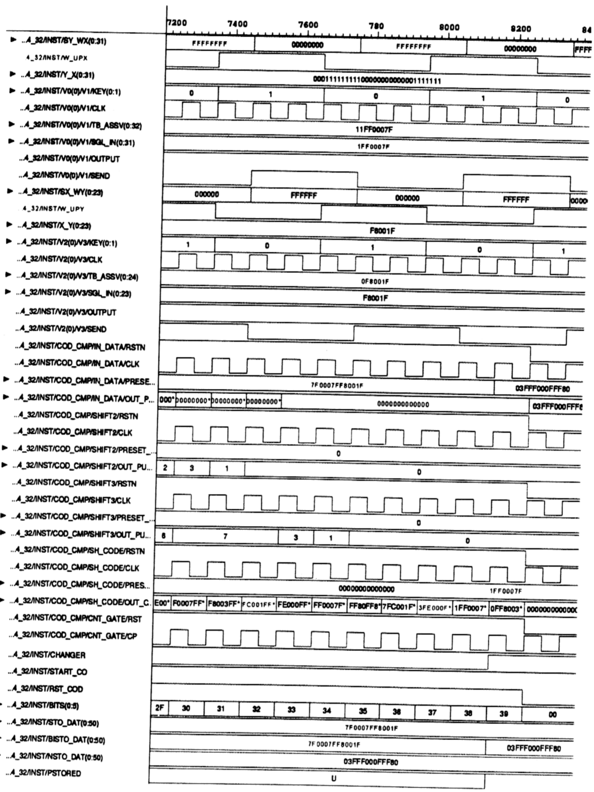

Figure (13). VHDL simulation Diagram of the Cyclic Encoder C (51,56) injected in BAM circuit (24 Neurons/ 32 Neurons) in the time interval [7200 ns, 8400 ns].

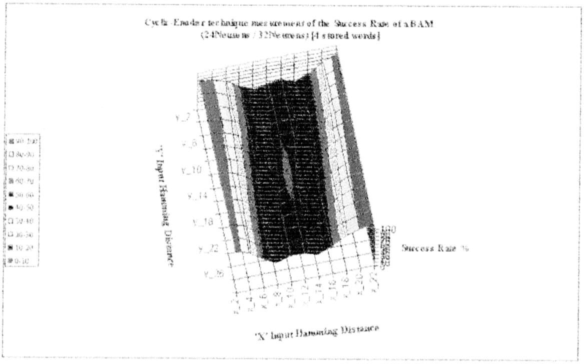

Figure (14). (EC) Technique Success rate measurement (4 stored memories)

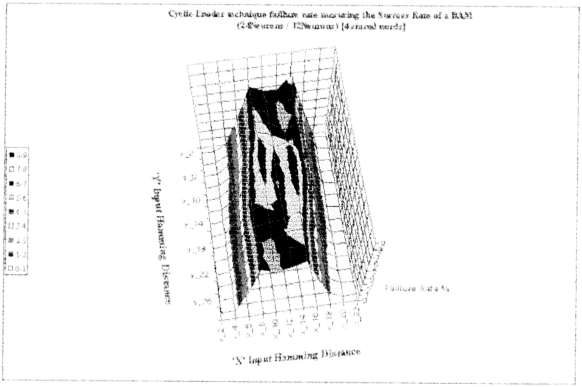

Figure (15). (EC) Technique failure rate measuring the success rate (4 stored memories)

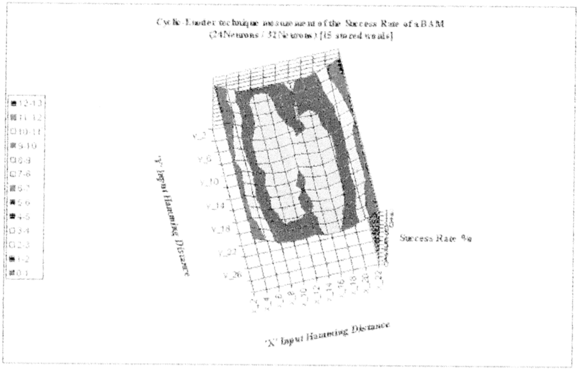

Figure (16). (EC) Technique Success rate measurement (15 stored memories)

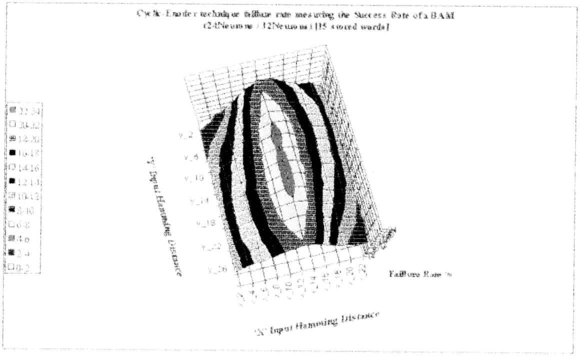

Figure (17). (EC) Technique failure rates measuring the success rate (15 stored memories).

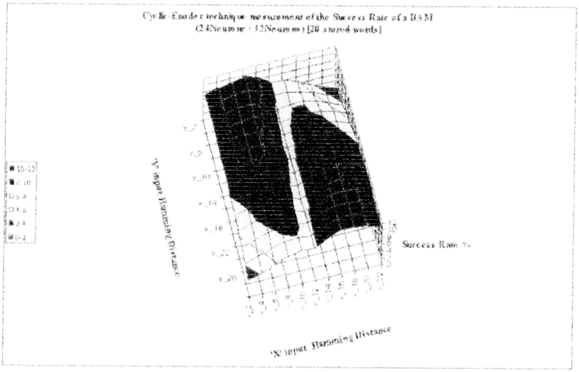

Figure (18) (EC) Technique Success rate measurement (20 stored memories)

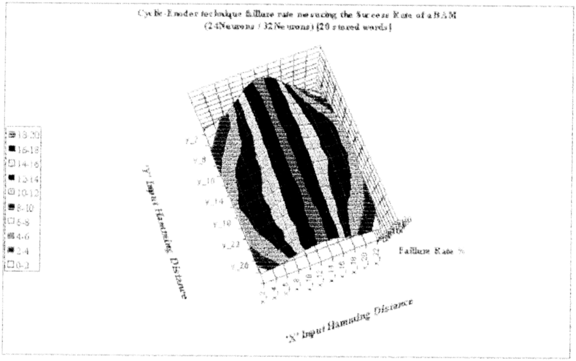

Figure (19). (EC) Technique failure rate measuring the success rate (20 stored memories)

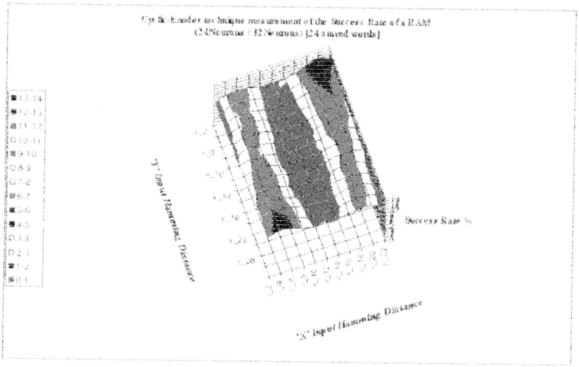

Figure (20). (EC) Technique Success rate measurement (24 stored memories)

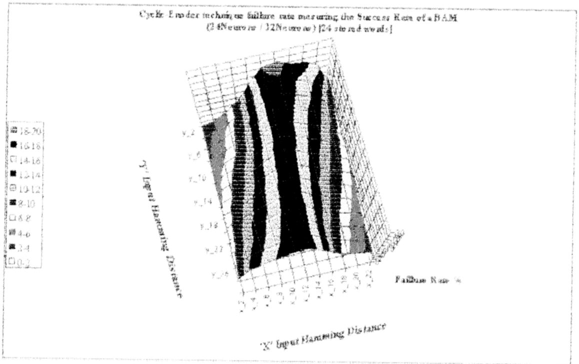

Figure (21). (EC) Technique failure rate measuring the success rate (24 stored memories)