A VLSI Architecture for Fast Clustering with Fuzzy ART Neural Networks

E. Granger[†], Y. Blaquire[‡], Y. Savaria[†], M.-A. Cantin[‡], and P. Lavoie[§] [*†]

[†] Electrical and Computer Engineering Dept., cole Polytechnique de Montral, P.O. Box 6079, station “Centre-ville”, Montreal, Qc., Canada, H3C 3A7, egranger@grm94.polymtl.ca.
[‡] Computer Science Dept., Universit du Qubec Montral, P.O. Box 8888, station A, Montreal, Qc., Canada, H3C 3P8, Blaquiere.Yves@uqam.ca.
[§] Defence Research Establishment Ottawa, Department of National Defence, Canada, K1A 0Z4, Pierre.Lavoie@dreo.dnd.ca.

Abstract

Hardware implementation of the Fuzzy ART neural network applied to a demanding real time radar signal clustering problem is investigated. To obtain efficient solutions for the implementation of this neural network in dedicated digital VLSI hardware, the network's algorithm is reformulated. A novel Fuzzy ART system architecture which can implement this reformulated algorithm for high speed clustering problems is then proposed. This system architecture is composed of a global comparator and several identical elementary modules (EMs), each of which emulates a number of neurons. The general architecture of each EM consists of a local comparator, dividers, neural processors, and a block of memory. This paper also outlines a cost analysis model for the estimation of area and processing time required with the proposed system architecture. This model offers a framework for rapid estimation of area-time complexity, given a VLSI system architecture configuration and a set of application constraints. As an illustration, this model is employed to estimate the performance of Fuzzy ART hardware applied to the clustering of radar pulses in electronic warfare systems.

[*] The authors would like to acknowledge the contribution of Eric Audet.[‡]

[†] This work was supported in part by the Defence Research Establishment Ottawa, the Canadian Microelectronics Corporation, le Fonds pour la Formation de Chercheurs et l'Aide la Recherche, and the Natural Sciences and Engineering Research Council of Canada.

1: Introduction

On-line clustering of intercepted radar pulses is important for data reduction in electronic support measures (ESM) systems. The purpose of radar ESM is to search for, intercept, locate, and analyze radar signals in the context of military surveillance. It is an important preliminary step in all electronic warfare systems, which allows the evaluation of the countermeasures to be undertaken for self-protection [1]. The need for data reduction in radar ESM arises from the signal densities encountered in some theaters of operation, which can reach up to 10^6 radar pulses per second. Grouping radar pulses into categories corresponding to active radar emitters at the early stages of processing could considerably reduce the processing requirements, as well as the hardware cost of ESM systems [2] [3] [4]. Since the number and description of the radar emitters are *a priori* unknown, the radar pulses must be grouped into categories based on their perceived similarity, a process which is called clustering [5] [6]. This process is hindered by the fact that pulses from one emitter are different from each other because of noise, and because of receiver, environmental and emitter variability [4].

In this paper, a neural network called Fuzzy ART [7] is considered for clustering. The network is capable of fast, stable, and unsupervised learning of categories in response to non-stationary sources of arbitrary binary or analog patterns. This network is attractive for ESM applications, since it can cluster patterns autonomously, in real-time, without prior knowledge of the categories.

This paper presents a flexible and modular digital VLSI architecture for Fuzzy ART neural networks suitable for very high data rate clustering applications. Flexibility and modularity allow adjusting system parameters to trade off area and speed. Also, due to the modularity, systems derived from the proposed architecture can be partitioned into several identical modules, which can be implemented in individually packaged chips, or as separate semiconductor dies interconnected in a multi-chip package. An area-time estimation model is provided to ease parameter selection, and to assess achievable semiconductor areas and data rates. Radar ESM is used as a case study to demonstrate the architecture and the use of this estimation model.

The paper is organized as follows. Fuzzy ART is reviewed as a neural network and then as an algorithm in the next section. In Section 3, the algorithm is reformulated to permit more efficient implementation without entailing a clustering performance degradation. In Section 4, a multi-chip VLSI system architecture based on the reformulated algorithm is proposed. Finally, Section 5 outlines a model for the rapid estimation of semiconductor area and processing time, given a VLSI system architecture configuration and a set of application constraints.

2: Fuzzy ART neural network

Adaptive Resonance Theory (ART) neural networks [8] can develop stable recognition capability on-line by self-organization, in response to arbitrary sequences of input patterns. Several different neural models based on ART have been proposed:

ART1, ART2, ARTMAP, ART3, etc. [9]. The Fuzzy ART neural network proposed by Carpenter, Grossberg and Rosen [7] introduces modifications to ART1 (replaces crisp logic AND operators (intersection: $\bigcap$) with fuzzy logic AND operators (minimum: $\wedge$)), which allow for processing of analog input patterns, as well as binary ones.

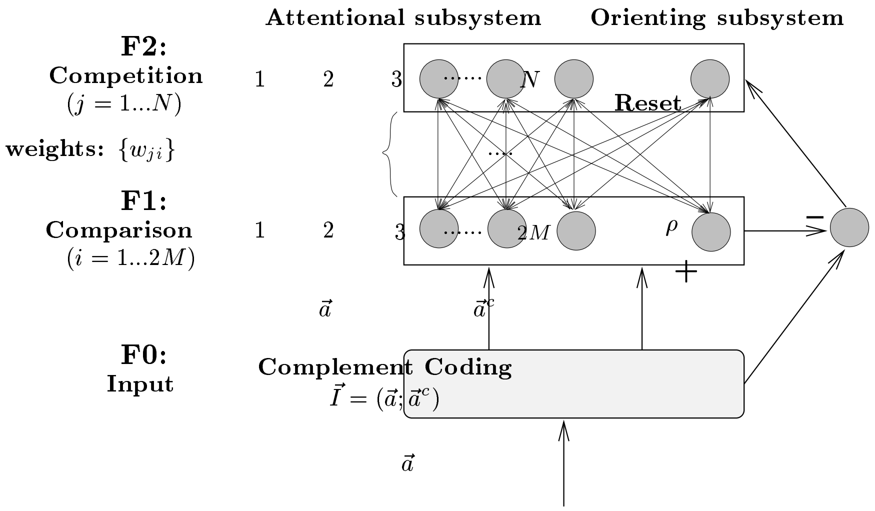

Figure 1. The Fuzzy ART neural network.

2.1: Neural network model

The general structure of the Fuzzy ART neural network is shown in Fig. 1. It consists of two layers of neurons that are fully connected: a $2M$ neuron input or *comparison* layer (F1) and an N neuron output or *competitive* layer (F2). A weight value w_{ji} is associated with each connection, where the indices i and j denote the neurons that belong to the layers F1 and F2 respectively. The set of weights $\vec{W} = \{w_{ji} : i = 1, 2, ..., 2M; j = 1, 2, ..., N\}$ encodes information that defines the categories learned by the network. These can be modified dynamically during network operation. For each neuron j of F2, the vector of adaptive weights $\vec{w}_j = (w_{j1}, w_{j2}, ..., w_{j2M})$ corresponds to the subset of weights ($\vec{w}_j \subset \vec{W}$) connected to neuron j. This vector $\vec{w}_j$ is named *prototype vector*, and it represents the set of characteristics defining the category j. Each prototype vector $\vec{w}_j$ is formed by the characteristics of the input patterns to which category j has previously been assigned through *winner-take-all* competitions. Fig. 1 shows that the Fuzzy ART neural network dynamics are governed by 2 subsystems. The *attentional subsystem* is responsible for proposing a winning category neuron, whereas the *orienting subsystem* accepts the proposed candidate or else reorients the search. Further details on these 2 subsystems, as well as on various gain control signals can be found in the literature [8], and are not discussed in this work.

2.2: Algorithmic description of Fuzzy ART

The Fuzzy ART network's functionality may be described by an algorithm [10] (as shown in Fig. 2). This algorithm can be divided into five execution steps:

1. **Weights and parameters initialization:** Initially, all the neurons of F2 are uncommitted, and all weight values w_{ji} are initialized to 1. An F2 neuron becomes committed when it is selected for an input $\vec{a}$. Then, the corresponding weights w_{ji} can take values expressed by a real number in the interval [0,1].
2. **Input vector coding:** When a new input vector $\vec{a} = (a_1, a_2, ..., a_M)$ of M elements (where each element a_i is a real number in the interval [0,1]) is presented to the network, it undergoes a preliminary coding at layer F0. *Complement coding* of $\vec{a}$ results in a network input vector $\vec{I}$ of $2M$ elements such that: $\vec{I} = (\vec{a}; \vec{a}^c) = (a_1, a_2, ..., a_M; a_1^c, a_2^c, ..., a_M^c)$, with $a_i^c = 1 - a_i$. This coding is recommended [7] to prevent a category proliferation problem that may occur in analog ART networks [10].
3. **Category choice:** With each presentation of an input $\vec{I}$ to F1, the *choice function* $T_j(\vec{I})$ is calculated for each neuron j in F2:

$$T_j(\vec{I}) = \frac{|\vec{I} \wedge \vec{w}_j|}{\alpha + |\vec{w}_j|} \tag{1}$$

   where $|\cdot|$ is the norm operator ($|\vec{x}| = \sum_{i=1}^{2M} |x_i|$), $\wedge$ is the fuzzy logic AND operator ($\vec{I} \wedge \vec{w}j = (\min(I_1, w_{j,1}), \min(I_2, w_{j,2}), ..., \min(I_{2M}, w_{j,2M}))$), and α is a user-defined *choice parameter* such that $\alpha > 0$. F2 is a *winner-take-all* competitive layer, where the winner is the neuron $j = J$ with the greatest value of activation T_j for the input $\vec{I}$ ($T_J = \max\{T_j : j = 1...N\}$). If the same T_J value is obtained by two or more neurons, the one with the smallest index j wins. The winning neuron J is retained for Steps 4 and 5.
4. **Vigilance test:** This step serves to compare the similarity between the prototype vector of the winning neuron $\vec{w}_J$ and input $\vec{I}$, against a user-defined *vigilance parameter* ρ, through the following test:

$$\frac{|\vec{I} \wedge \vec{w}_J|}{|\vec{I}|} \geq \rho \tag{2}$$

   where $\rho = [0, 1]$. This comparison is carried out on layer F1: the winning neuron J transmits its learned expectancy, $\vec{w}_J$, to F1 for comparison with $\vec{I}$. If the vigilance test (Eq. 2) is passed, then neuron J becomes *selected* and is allowed to adapt its prototype vector (Step 5). Otherwise, neuron J is deactivated for the current input $\vec{I}$: T_J is set equal to -1 for the duration of the current input presentation. The algorithm searches through the remaining F2 layer neurons (Steps 3 and 4), until some other neuron J passes the vigilance test. If no committed neuron from the F2 layer can pass this test, an uncommitted neuron is selected and undergoes prototype vector update.

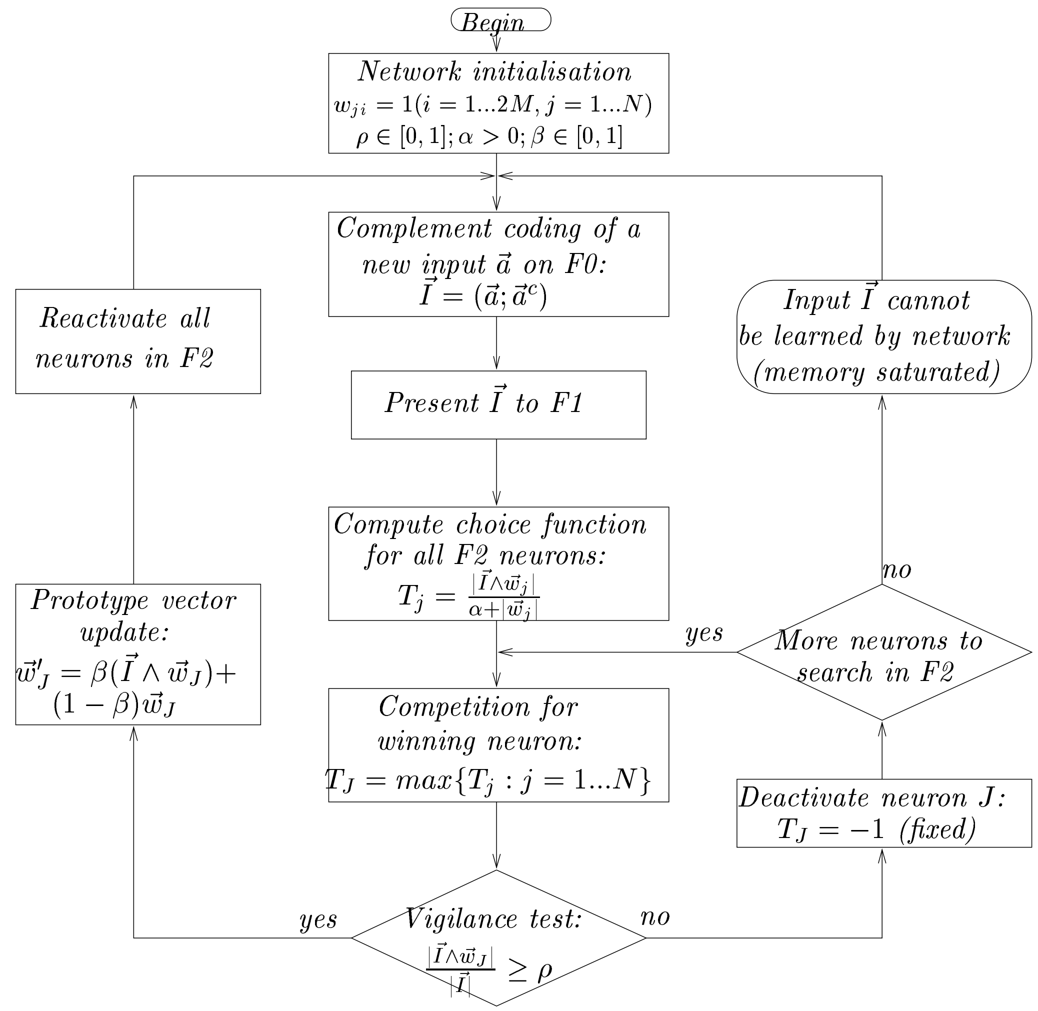

Figure 2. Flowchart representation of the Fuzzy ART algorithm.

5. **Prototype vector update:** The prototype vector of the winning neuron J is updated according to:

$$\vec{w}'_J = \beta(\vec{I} \wedge \vec{w}_J) + (1-\beta)\vec{w}_J \tag{3}$$

where β is a user-defined *learning rate parameter* such that $\beta = [0,1]$. The algorithm can be set to slow learning, with $0 < \beta < 1$, or to fast learning, with $\beta = 1$. Once this update step is accomplished, the network can process a new input vector (Step 2).

The flowchart representation in Figure 2 summarizes the Fuzzy ART algorithm. It is assumed that the number of neurons N in layer F2 is fixed but arbitrary.

The Fuzzy ART algorithm was simulated with real radar signal data. The data set used was collected by the Defence Research Establishment Ottawa, and it contains 800 radar pulses from 12 different emitters. Each radar pulse in this set is characterized by 8 complex features. The network inputs $\vec{a}$ therefore have $M = 16$ elements in the interval [0,1]. Based on these functional simulations, the following set of network parameters: $\beta = 1$, $\alpha = 1$ and $\rho = [0.75, 0.9]$ was determined. These values correspond to a satisfactory trade-off between clustering quality and algorithm simplicity. Setting $\beta = 1$ in Eq. 3 allows for fast learning with a simple prototype vector update equation. Considering the intended application, and the simulation results, it was decided that the network would only be implemented for the fast learning case. Moreover, since our objective is to obtain an effective, resolution-limited digital hardware implementation, it was also determined that the minimum word length b required to represent the values w_{ji}, I_i, and ρ is $b = 11$ bits. This produces errors within 2% of the clustering results obtained with floating-point values.

2.3: Hardware realizations of ART neural networks

The authors could find no reference to previous efforts to realize the Fuzzy ART neural network in hardware. However, several attempts were made to implement the ART1 [8] (binary input) neural network and, as mentioned earlier, ART1 and Fuzzy ART processing is very similar. Different classes of ART1 realizations have been reported. The first class fully or partially implements the time-domain nonlinear differential equations that were originally used to describe ART1 (for example, [11] - [13]). Indeed, according to the original modeling of ART1, a set of differential equations describe the dynamics of each neuron and synapse [8]. This network formulation and the derived realizations demand a great deal of computational power. The second class of realization consists in replacing the differential equations of the original model with a set of steady-state nonlinear algebraic equations, which can be executed sequentially to produce the exact same result (for example, [14] - [18] for ART1, and [19] for ART2). These algorithmic descriptions require artificial sequencing of the events in the original model, but they are less computationally expensive, allowing for a higher potential throughput. Some of the implementations are based on analog, or mixed analog/digital circuitry (for example [11], [12], [13], [17]). Others are optical based implementations (for example [14], [18], [19]). Yet others are realized as software executing on existing hardware (for example [20]).

Our goal is to realize a class 2 implementation of Fuzzy ART with a dedicated digital VLSI system for maximum throughput with a high degree of accuracy. Rao et al. [15] [16] have reported on a class 2 ART1 network implementation based on digital hardware. The authors exploit a multiplexing technique that permits simultaneous matching of inputs to prototype vectors, and generation of F2 activation values. They reduce ART1's processing to a linear search, best match algorithm. The search's functionality is implemented with a pipelined associative memory architecture: a pipeline of identical, back-to-back processing elements, capable of simultaneously matching several inputs against stored prototype vectors. The architecture achieves a temporal parallelism (since many inputs can be processed at once), which results in a very high throughput for small scale problems (16 possible categories, 9 bit input vector, etc.). This architecture is not appropriate for the target ESM applications where the number of categories is in the hundreds, and where input and prototype vectors contain $2M \times b = 32$ elements $\times$ 11 bits = 352 bits. Moreover, the architecture is somewhat incomplete since it lacks the mechanism needed for the update of already committed categories. These shortcomings have motivated our desire to propose a new system architecture capable of high throughput with a minimum dependence on the number of categories, which incorporates all features of Fuzzy ART processing.

3: Reformulated Fuzzy ART algorithm

It would be possible to directly implement the Fuzzy ART neural network's algorithm as described in Figs. 1 and 2. However, neuron interconnections would occupy excessive semiconductor area ($O(M \cdot N)$), and would eventually limit the size of a feasible digital hardware implementation of the network. Furthermore, simulations with real radar data have shown that a large proportion of the prototype vectors remain unchanged during the update operations. Prior to its implementation in dedicated hardware, the Fuzzy ART algorithm (given in Fig. 2) was reformulated to circumvent its massively parallel processing requirements, and to exploit its inherent potential for re-use of data. The reformulated Fuzzy ART algorithm helps to maximize throughput, to minimize communication costs, and is easier to implement in digital VLSI hardware for high throughput applications.

Although the reformulated algorithm is functionally equivalent to the original one, it is expressed as a stream of operations that can be carried out sequentially. Other modifications to the original algorithm are based on observations of the Fuzzy ART algorithm. Computation of the choice functions (a division, T_j, for every committed neuron j), and their components ($|\vec{I} \wedge \vec{w}_j|$ and $|\vec{w}_j|$), constitutes a substantial proportion of the total processing effort. Fortunately, once the elements $|\vec{I} \wedge \vec{w}_j|$ and $|\vec{w}_j|$ have been computed, their values can be re-used on several occasions. This permits a more efficient organization of the algorithm's processing. In addition, there is no need to compute the choice function T_j of a neuron j that would not pass the vigilance test. This modification circumvents the original algorithm's search process, since the neuron $j = J$ with the greatest T_j value that can pass the vigilance test is readily selected. The vigilance test should thus occur as soon as the component $|\vec{I} \wedge \vec{w}_j|$ is available for a neuron j. Further, since complement coding is an input normalization such that, by definition, $|\vec{I}| = |(\vec{a}; \vec{a}^c)| = M$ [7], the vigilance test reduces to $|\vec{I} \wedge \vec{w}_j| \geq \rho \cdot M$. In parallel with the vigilance test, it is convenient to carry out a *subset test*, since it depends only on the elements $|\vec{I} \wedge \vec{w}_j|$ and $|\vec{w}_j|$. This test detects when a prototype vector $\vec{w}_j$ is a subset of $\vec{I}$, that is, when $|\vec{I} \wedge \vec{w}_j| = |\vec{w}_j|$. If this neuron j is chosen and passes the vigilance test, then its prototype vector $\vec{w}_J$ remains unchanged during the learning phase, since $\vec{w}'_J = \vec{I} \wedge \vec{w}_J = \vec{w}_J$ in fast learning mode ($\beta = 1$). Therefore, the learning phase can be bypassed, and the network speed increased.

These observations lead to a reformulation of the Fuzzy ART algorithm, as shown in Fig. 3 for the fast learning case. The algorithm sequentially processes committed neurons only, and N now represents the number of committed neurons, which starts from 1 and increases progressively as learning takes place. The reformulated algorithm is divided into two parts: a *recall phase* and a *learning phase*. During the recall phase, the values T_j and S_j are computed for all committed neurons. If a neuron j is a subset choice of $\vec{I}$, then S_j is set equal to 1. If a neuron j passes the vigilance test for $\vec{I}$, then the choice function is computed and $T_j > 0$, otherwise T_j is set equal to 0. At the end, the winning neuron J is chosen. During the learning phase, in the case where $T_J \geq \frac{M}{\alpha+2M)}$, the winning neuron J corresponds to a committed category that passes the vigilance test. If J is a subset choice for $\vec{I}$ ($S_J = 1$), then the prototype vector update is bypassed; otherwise ($S_J = 0$), the prototype vector $\vec{w}_J$ is updated according to Eq. 3. In the case where $T_J < \frac{M}{\alpha+2M)}$, no committed category has passed the vigilance test. If the maximum number of neurons is not attained ($N < N_{tot}$), then an uncommitted neuron $J = N + 1$ is assigned to $\vec{I}$; otherwise ($N = N_{tot}$), the network is unable to categorize $\vec{I}$ because its memory is saturated. After the learning phase, the network is ready to accept another input vector.

4: Fuzzy ART system architecture

In this section, we present a dedicated VLSI system architecture for the reformulated Fuzzy ART algorithm of Fig. 3. Our main objective is to execute this algorithm at high speeds. It is true that analog implementations can be more compact (since neuron interconnections consume less semiconductor area), and have greater potential for high speed processing. However, digital implementations offer higher computational accuracy, more flexibility in design, better resistance to noise and interference, more reliable testability methods, and less susceptibility to process variations among devices [21]. It is prohibitively costly or not currently feasible to construct a relatively large massively parallel digital network on a single integrated circuit, as with the analog case. Therefore, any attempt to achieve a high degree of parallelism results in very large circuits, whose area is dominated by interconnections [15]. Given the nature of the application, a multi-chip digital implementation was selected in order to ensure that the required accuracy can be maintained.

The actual number of neurons is not necessarily the predominant constraint when considering a digital system implementation, since a neural processor (NP) can sequentially process, and thus emulate, many neurons. Yet, the system performance deteriorates as the number of neurons emulated per NP grows (due to increased processing time and memory size). The parallel nature of the intensive computations carried out by this neural network means that they execute very slowly on sequential processor architectures [22]. The potential for success in terms of throughput of the digital system approach thus highly depends on the feasibility of the required level of parallel execution [23].

Bearing in mind ESM applications, we seek a Fuzzy ART network with the following parameters: $2M = 32$ neurons in the F1 layer, $N_{tot} \geq 250$ category neurons in the F2 layer, fast learning ($\beta = 1$), and $b = 11$ bit word length. For proof-of-concept, the Fuzzy ART network should accept and categorize a new input vector every 2μs (500 000 patterns per second). Given the constraints imposed by current VLSI technology, the high speed processing requirements, and the number of category neurons needed, a dedicated digital VLSI hardware implementation of the network's

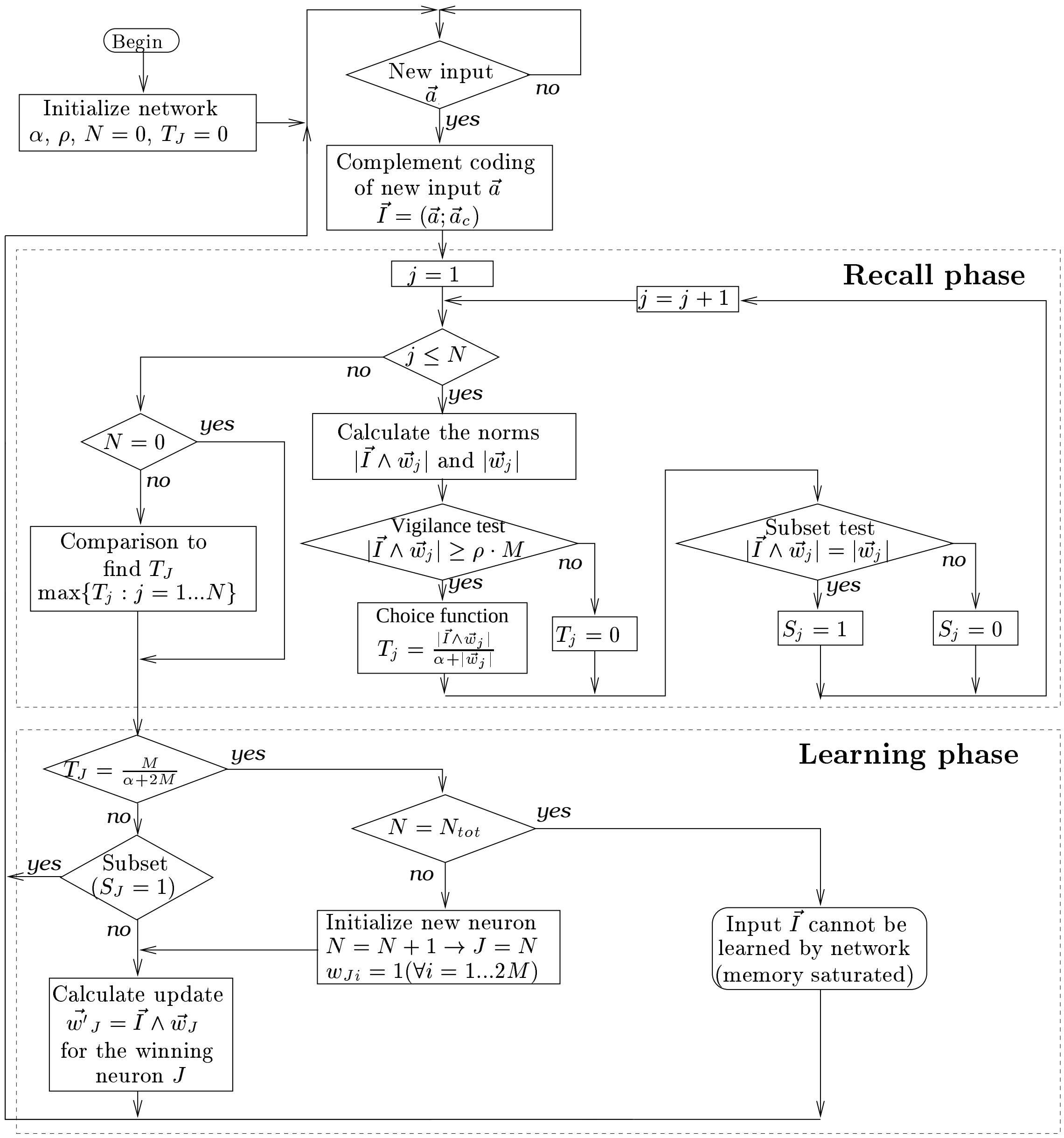

Figure 3. Flowchart representation of the reformulated Fuzzy ART algorithm.

processing should preferably be distributed over several integrated circuits. For the remainder of this section, a multi-chip Fuzzy ART system architecture is presented. In the next section, a detailed area-time estimation shows the performance achievable with the proposed system.

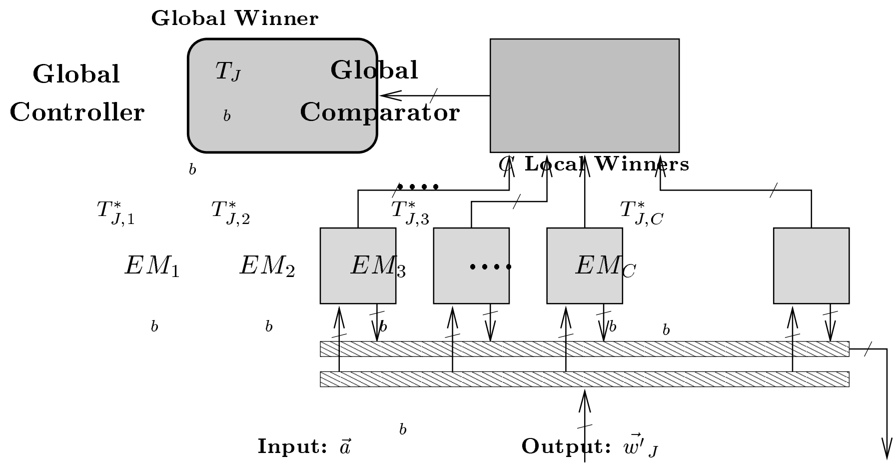

Figure 4. Architecture of Fuzzy ART system.

4.1: Proposed System Architecture

The proposed Fuzzy ART system architecture (Fig. 4) is composed of a global comparator, a global controller, and C identical elementary modules (EMs), each of which emulates $N = \frac{N_{tot}}{C}$ neurons. The C EMs can be implemented either in individually packaged chips, or as separate semiconductor dies interconnected together in a multi-chip package. Each EM determines the T^*_J of its local winner J^*, for an input $\vec{a}$ of M elements, considering the N neurons it emulates. The global comparator then selects the global winner J, and enables the corresponding EM for update. The general data path architecture of an EM (shown in Fig. 5) consists essentially of a local controller, a local comparator, D fixed-point dividers, P neural processors (NPs) - each emulating $\frac{N}{P}$ category neurons -, $\frac{P}{D}$ NPs per divider, and D blocks of random access memory (RAM) for storage of the N prototype vectors.

To reduce the control burden, an entire EM is considered to be committed if any one of the N neurons it emulates is committed. Recall that uncommitted neurons have been initialized such that $|\vec{w}_j| = 2M$ (since $w_{ji} = 1\ \forall j, i$). When needed, a new EM is activated, and computes its local T^*_J based on the results from all its internal neurons, even though these neurons may not all be committed. This emulates Fuzzy ART's orderly selection of uncommitted neurons: the first uncommitted neuron in an EM is selected only when $T_j < \frac{M}{\alpha+2M}$ for all committed neurons ($\forall j = 1, 2, ..., N$). That is, according to Eq. 1, an uncommitted neuron cannot be selected unless all committed neurons have failed the vigilance test or less active than $M/(\alpha + 2M)$.

The speed of such a system depends mostly on the number of neurons emulated, and on access time to the prototype vectors [21]. Storage of 352-bit prototype vectors for $N_{tot} \cong 250$ neurons requires about 88$k$bits of RAM. In general, internal memory

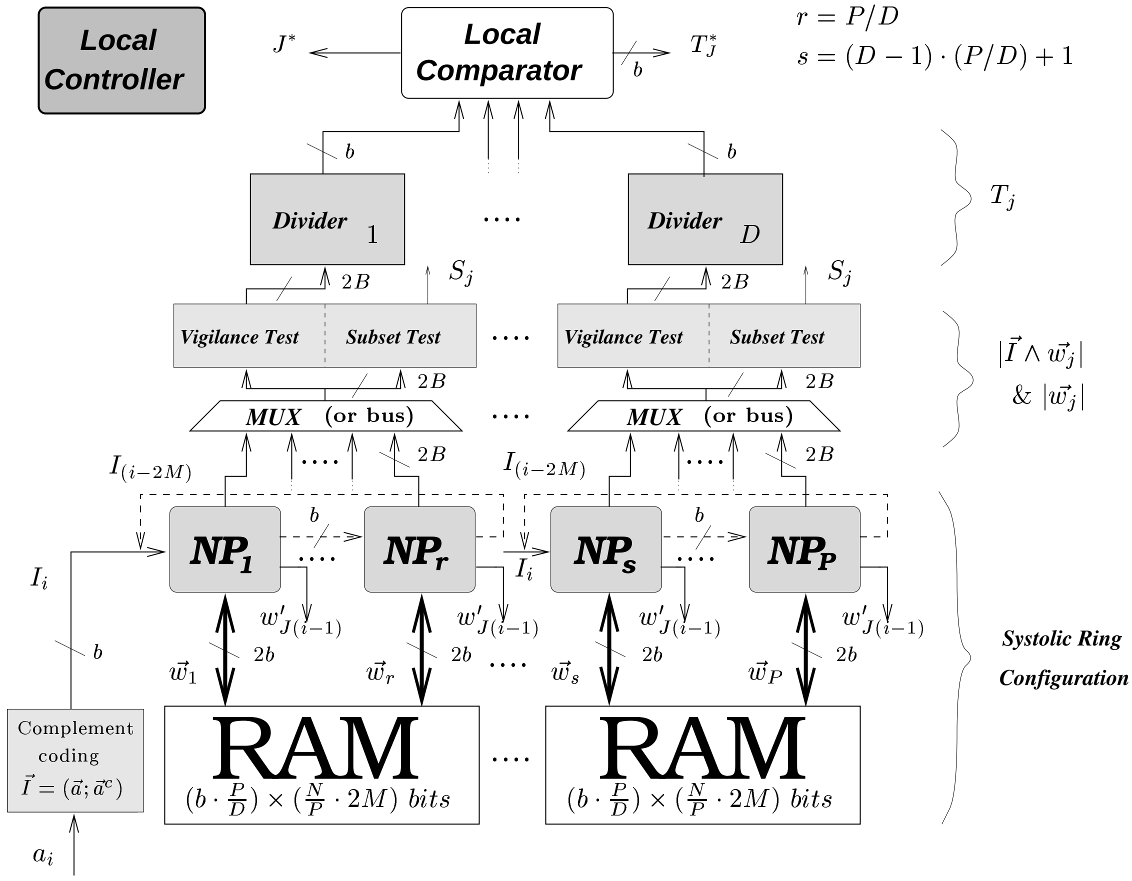

Figure 5. Architecture of an elementary module (EM).

is prefered for fast access to weights and moderate I/O pin count, whereas external memory is prefered to obtain a maximum number of emulated neurons. Considering that memory is only one part of the system, for a proof-of-concept implementation with the desired number of neurons, it would be preferable to use a cascade of several identical chips with their own internal RAM. If there are C EMs, then each EM stores the weights of $N = \frac{N_{tot}}{C}$ neurons (with an area $N \cdot 2M \cdot b \cdot A_{membit}$); and if there are P NPs per EM, then the memory associated with each NP contains the weights of $\frac{N}{P}$ neurons. On-chip storage permits very fast processing distributed over C identical EMs with moderate I/O bandwidth requirements. This also allows to balance areas devoted to memory and to functional circuit [21].

Fig. 6 presents the basic structure of a NP module. In agreement with the reformulated algorithm, a NP can operate in the recall phase or the learning phase. During the recall phase, the value $|\vec{w}_j \wedge \vec{I}|$ is calculated for every committed F2 neuron j, and during the learning phase, the prototype vector $\vec{w'}_J$ (and $|\vec{w'}_J|$) of the globally winning neuron is updated. Therefore, the system architecture performs the recall and learning phases of the reformulated Fuzzy ART algorithm sequentially using the same hardware. The system's behavior is described for both of these phases in the following subsections.

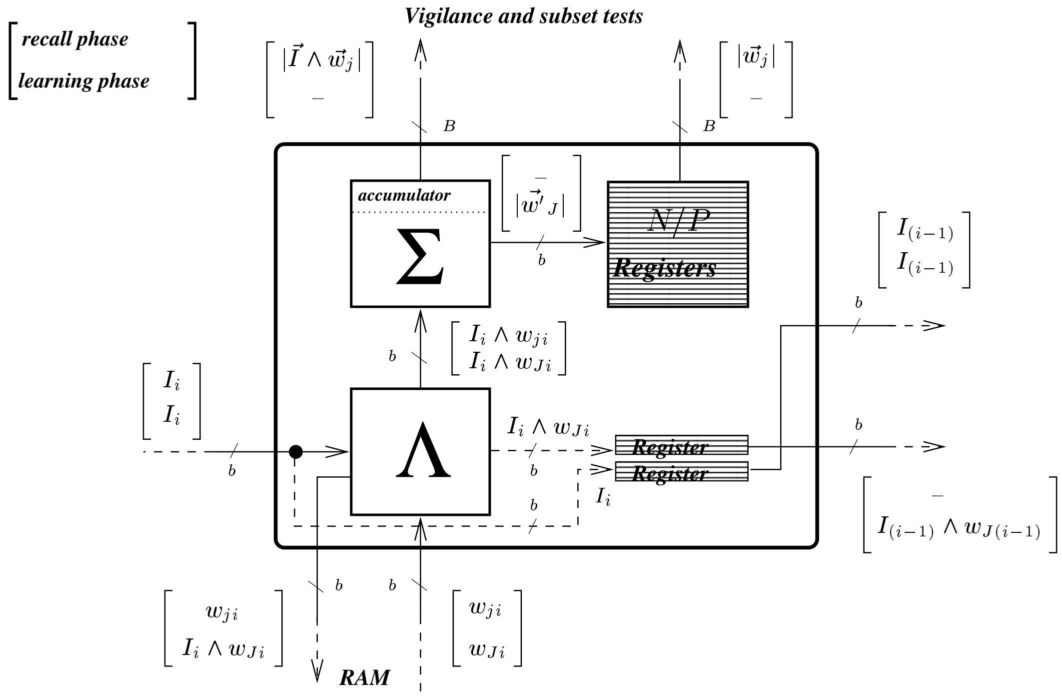

Figure 6. The basic structure of a NP module consists of a "min" comparator ($\wedge$), an adder-accumulator ($\sum$), and registers to compute the values $|\vec{I} \wedge \vec{w}_j|$, $\vec{w}'_J$ and $|\vec{w}'_j|$. Note the dual labels for recall (top) and learning (bottom) phases respectively.

4.2: Recall phase

Complement coding is done inside each EM circuit to reduce I/O requirements since it is a relatively simple operation. This coding has little impact on the system throughput, since the sequential complement coding of the M elements of input vector $\vec{a}$ overlaps the NPs processing. Accordingly, the delay is equal to t_{cc} and corresponds to the complement coding of the first element of $\vec{a}$. As an input $\vec{a}$ is being encoded inside an EM, the recall phase forms a data processing pipeline starting from the systolic ring configuration, and ending with the global comparators.

A *systolic ring configuration* [22] [24] is proposed for the NPs, to exploit the repetitive and regular nature of the $|\vec{I} \wedge \vec{w}_j|$ computation, and to increase the potential for parallel execution with VLSI technology. This configuration pipelines processing through a chain of locally connected NPs linked with unidirectional connections (see Figs. 5 and 7). The shifting of data between NPs offers a high total inter-NP throughput by using local communications only, yielding a better balance between communications and computations [23]. Further, such regular communications minimize I/O requirements, which allows to implement a large number of NPs per EM. It also simplifies the connections between each neuron and the rest of the circuit. Each systolic ring is connected to one divider.

Every NP occupies an area A_{NP}, and communicates with a local memory that contains the prototype vectors of $\frac{N}{P}$ category neurons of the F2 layer. The $2M$

elements of the complement coded input vector $\vec{I}$ are shifted right, in natural order sequence (I_1, I_2, etc.), through the NPs in a ring, while the $2M$ elements of every prototype vector, the weights, are sequentially read from RAM and shifted upwards. Respective input vector and weight elements for each neuron j are combined in a series of $2M$ minimum operations, $I_i \wedge w_{ji}$, so that each NP can accumulate its output value, $|\vec{I} \wedge \vec{w}_j|$. After $2M + (\frac{P}{D} - 1)$ cycles, all the output values of the systolic rings are ready.

Fig. 7 shows the interaction between $\vec{I}$ and the prototype vectors in one systolic ring. Tab. 1 shows an example of NP processing during the recall phase with $M = 16$, $\frac{P}{D} = 4$ and $\frac{N}{P} = 5$. Notice that for each input $\vec{I}$, $\frac{N}{P}$ cycles around each ring are required to process the entire set of emulated neurons. Fig. 6 shows that the value $|\vec{w}_j|$ (computed during the learning phase) is also generated from the NPs in the same sequence as its respective $|\vec{w}_j \wedge \vec{I}|$. The total time required to process the $\frac{N}{D}$ neurons associated with a divider D is no longer than $[2M(\frac{N}{P}) + (\frac{P}{D} - 1)] \cdot t_{rep}$. The systolic ring response time, t_{rep}, is defined by $t_{rep} = max\{t_{mem}, t_{NP}\}$, where t_{mem} is the memory access time and t_{NP} is the time needed to compute a "min" comparison $(I_i \wedge w_{ji})$ and an addition in a NP. For simplicity, we assume for the memory access time that $t_{mem} \simeq t_{\text{read}} \simeq t_{\text{write}}$. The factor $2M(\frac{N}{P})$ is the number of cycles required to compute the norms $|\vec{I} \wedge \vec{w}_j|$ in the NPs for $\frac{N}{D}$ neurons, whereas $\frac{P}{D} - 1$ is the latency for filling the pipe of the D systolic rings. We suppose that $1 \leq \frac{P}{D} \leq 2M$, thus the divider used cannot accept results faster than they are produced by the ring of NPs. The block of memory required for each set of $\frac{P}{D}$ NPs linked to a divider is $2M \cdot \frac{N}{P}$ words deep, each of which contains $\frac{P}{D}$ elements of b bits wide. Each NP requires a datapath with a minimum width of $B = b + \log_2(2M)$ bits to prevent overflows with the norm values $|\vec{I} \wedge \vec{w}_j|$ and $|\vec{w}_j|$, since there are $2M$ accumulate operations to be done on b bit words. Considering the relative simplicity of the NPs, and the sequential nature of the processing they perform, several NPs can be assigned to a divider (Fig. 5). Globally broadcasted signals to each NP control the sequence of operations in the ring.

The values $|\vec{w}_j \wedge \vec{I}|$ and $|\vec{w}_j|$ are transferred from each NP to its respective divider, after passing through vigilance and subset tests, which can conveniently be carried out in parallel. If $t_{rep} \geq t_{comp} = \max\{t_{vig}, t_{sub}\}$, the delay needed for these parallel tests (t_{vig} and t_{sub}) can be absorbed as a simple overhead cycle in the data processing pipe. Each vigilance test module contains the value $\rho \cdot M$ needed to perform the required comparison. It is possible, during processing, to adjust the value ρ in the range $[0, 1]$ to adapt category discrimination as a function of the input environment. If the value $|\vec{w}_j \wedge \vec{I}|$ of a neuron j passes the vigilance test, then this value is passed on to the divider; otherwise, T_j is set to 0, eliminating j from the competition. The corresponding value S_j is retained for use during the learning phase. The results of the subset test for an entire EM can be stored, for example, in the local controller with D registers of $\frac{N}{D}$ bits each.

Fixed-point dividers are used to compute the choice functions from the numerator $|\vec{w}_j \wedge \vec{I}|$ and the denominator $(|\vec{w}_j| + \alpha)$. Note that the value $(|\vec{w}_j| + \alpha)$ is computed as a preliminary step in parallel with the subset and vigilance tests. The numerator

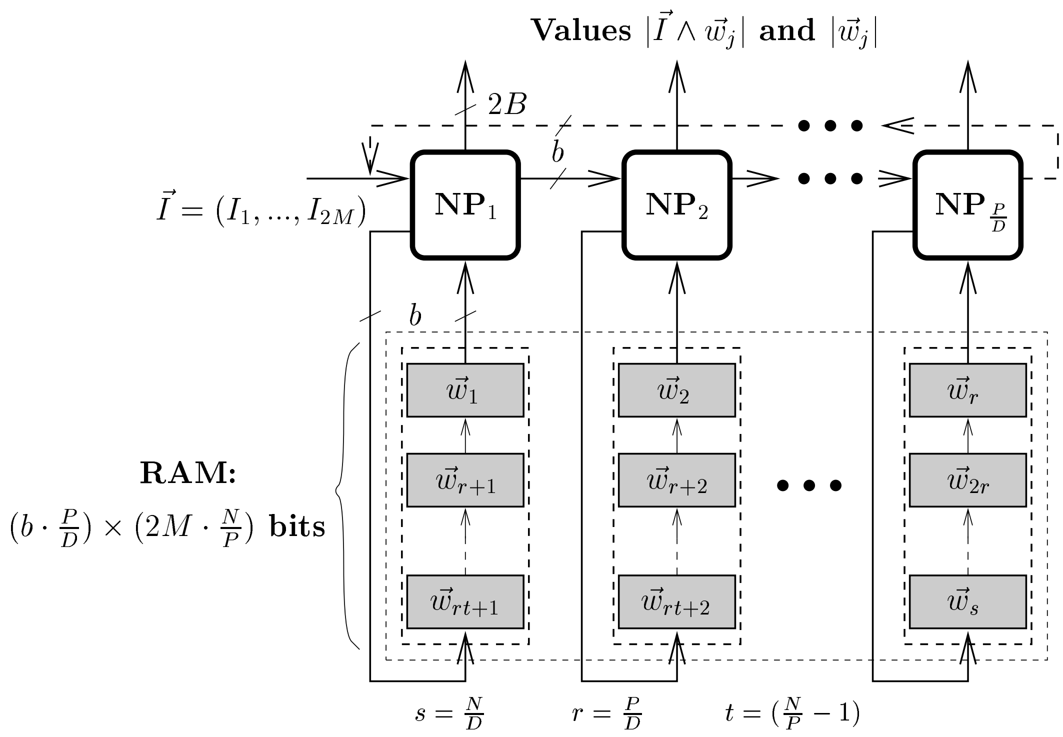

Figure 7. Systolic ring configuration used with NPs.

Table 1. Example of NP allocations in a systolic ring during the recall phase with $M = 16$, $\frac{P}{D} = 4$ and $\frac{N}{P} = 5$ (a block of $\frac{N}{D} = 20$ neurons). The results $|\vec{w}_j \wedge \vec{I}|$ are sequentially computed in the NPs as the inputs I_i and w_{ji} are shifted through. The values $|\vec{w}_j|$ (computed during the learning phase) are also generated by the NPs in the same sequence as their respective $|\vec{w}_j \wedge \vec{I}|$.

Time	**Inputs $\vec{I}$**				**Inputs $\vec{w}_j$**				**Outputs $\vert\vec{I} \wedge \vec{w}_j\vert$**			
(Cycles)	NP_1	NP_2	NP_3	NP_4	NP_1	NP_2	NP_3	NP_4	NP_1	NP_2	NP_3	NP_4
1	I_1				$w_{1,1}$							
2	I_2	I_1			$w_{1,2}$	$w_{2,1}$						
3	I_3	I_2	I_1		$w_{1,3}$	$w_{2,2}$	$w_{3,1}$					
4	I_4	I_3	I_2	I_1	$w_{1,4}$	$w_{2,3}$	$w_{3,2}$	$w_{4,1}$				
5	I_5	I_4	I_3	I_2	$w_{1,5}$	$w_{2,4}$	$w_{3,3}$	$w_{4,2}$				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(1 \cdot 2M) + 0$	I_1	I_{32}	I_{31}	I_{30}	$w_{5,1}$	$w_{2,32}$	$w_{3,31}$	$w_{4,30}$	$\vert \vec{I} \wedge \vec{w}_1 \vert$			
$(1 \cdot 2M) + 1$	I_2	I_1	I_{32}	I_{31}	$w_{5,2}$	$w_{6,1}$	$w_{3,32}$	$w_{4,31}$		$\vert \vec{I} \wedge \vec{w}_2 \vert$		
$(1 \cdot 2M) + 2$	I_3	I_2	I_1	I_{32}	$w_{5,3}$	$w_{6,2}$	$w_{7,1}$	$w_{4,32}$			$\vert \vec{I} \wedge \vec{w}_3 \vert$	
$(1 \cdot 2M) + 3$	I_4	I_3	I_2	I_1	$w_{5,4}$	$w_{6,3}$	$w_{7,2}$	$w_{8,1}$				$\vert \vec{I} \wedge \vec{w}_4 \vert$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(5 \cdot 2M) + 3$												$\vert \vec{I} \wedge \vec{w}_{20} \vert$

$|\vec{w}_j \wedge \vec{I}|$ and the denominator $(|\vec{w}_j| + \alpha)$ are represented with $B = b + \log_2(2M)$ and $B+1$ bits respectively, and the final result T_j is b bits wide.

The division efficiency is very important to the system's performance: it must be reasonably fast, without sacrificing the accuracy of the results. To maximize throughput, each divider (of delay t_D and area A_D) contains E pipeline stages (of delay $(t_{div} + t_{latch})$) to reduce the clock period. Evenly spread levels of pipelining registers (of delay t_{latch} and area A_{level}) ensure that the divider's critical path delay satisfies $(t_{div} + t_{latch}) = \frac{t_D}{E} \le t_{rep}$, where E is the number of pipeline stages. With such a pipelining, the NP results are fed sequentially to dividers having E pipeline levels, generating a result T_j after a delay of $t_D = E \cdot t_{rep}$.

Following the divisions, the resulting T_js are fed to a local comparator. The maximum T_j^* in the current cycle is compared to the maximum T_j^* of the previous cycles, until all the T_j^*s have been compared, and a *local winner* J^* has been found. The index J^* and the value T_J^* of the local winner are passed on to the local controller for global comparison, yielding a *global winner* J. The EM containing this global winner J is then activated for the learning phase. The local and global comparisons can be implemented using $\lceil \log_2(D) \rceil$ and $\lceil \log_2(C) \rceil$ level binary trees of $(D-1)$ and $(C-1)$ comparators, with a processing time that grows as $O(\log_2(D))$ and $O(\log_2(C))$ respectively. If comparison trees are used, the worst case processing delay for finding a local winner is $\lceil \log_2(D) \rceil \cdot t_{lcom}$, and for finding a global winner, $\lceil \log_2(C) \rceil \cdot t_{gcom}$.

4.3: Learning phase

Rather than recomputing the $|\vec{w}_j|$ for all the neurons in use at every recall phase, only the $|\vec{w}_J|$ of the updated neuron is recomputed, using the existing NP hardware. This $|\vec{w}_J|$ updating operation is done during the *learning phase*, while updating the actual $\vec{w}_J$. During this phase, the prototype vector $\vec{w}_J$ of the winning neuron is updated by its respective NP in $4M$ cycles: the weights w_{Ji} (with $i = 1, 2, ..., 2M$) are loaded from memory, processed in the NP, and then written back to memory. This update is bypassed when J is a subset choice for $\vec{I}$ (that is, if $S_j = 1$). The processing of the learning phase is similar to that of the recall phase, except that only the winning neuron's prototype elements are shifted upwards through the associated NP, in sequence with $\vec{I}$, to compute $\vec{w}'_J = \vec{w}_J \wedge \vec{I}$: all the other NPs simply shift the input through. At the same time, the elements w'_{Ji} are sequentially shifted outwards from the NP (as shown in Fig. 6), out of the EM, and then out of the system on a b bit bus (see Fig. 4). This allows external monitoring of new and updated categories, in real time. This feature is essential if the clustering results are to be communicated to the rest of the system for real time decision making.

For the learning phase, the architecture requires, at most, a delay of $t_{learn} = [(4M + \frac{P}{D} - 1)] \cdot t_{rep}$, where $(\frac{P}{D} - 1) \cdot t_{rep}$ is an additional latency due to input vector shifting through $\frac{P}{D}$ NPs (in the worst case, J is a multiple of $\frac{P}{D}$). To reduce the delays of this phase, the EMs can begin updating their local winners while the global comparison is being done. The local results $\vec{w}'_J$ can be stored temporarily. If the local winner is not the global winner, the update procedure is interrupted, and the local winner's weights remain unchanged. Otherwise (i.e. if the local winner is the global winner), the $\vec{w}'_J$, which is being held in a temporary buffer, can be transfered to replace $\vec{w}_J$. This minimizes the influence of the number of chips, C, on the system's performance, but entails additional EM controller complexity and memory.

5: Choice of system configuration

This section presents an area-time estimation model for the system architecture described above, and shows its application to a radar ESM problem. The development of this model was motivated by a desire to determine the impact of parameter choices on the system's processing rate and area. Given the performance constraints of an application, this model allows fast analysis of trade-offs, resulting in the selection of an appropriate system configuration.

5.1: Area-time estimation model

A cost function, $A_{tot} \cdot t_{tot}$, that we wish to minimize, is used to assess the system architecture considering a given set of parameters. This function is the product of A_{tot}, the area of the system's datapath, and t_{tot}, the time required to process an M element input vector $\vec{a}$.

The area A_{tot} corresponds to the sum of global and local comparator areas, divider areas, NP areas, and the area of the memory for prototype vectors. The interconnect area, and a few small modules are neglected. The equation for computing A_{tot} is:

$$A_{tot} = A_{\text{Global Comp.}} + C\left[A_{\text{Local Comp.}} + A_{\text{Dividers}} + A_{\text{NPs}} + A_{\text{Memory}}\right]$$

$$A_{tot} = (C-1)A_{gcom} + C[(D-1)A_{lcom} + D(A_D + EA_{level}) + PA_{NP} + 2MNbA_{membit}] \qquad (4)$$

The time t_{tot} is approximated by making abstraction of communication time and chip I/O delays. It is equal to the sum of the delays associated with the recall and learning phases: $t_{tot} = t_{recall} + t_{learn}$. The parameter t_{recall} consists essentially of the sum of the delays required for the norm computations in the NPs ($|\vec{I} \wedge \vec{w}_j|$) and the latency of the complement coding of one element of $\vec{a}$, one subset or vigilance test, one complete division, one local and one global comparison:

$$t_{recall} = t_{cc} + \left[2M\lceil\tfrac{N}{P}\rceil + (\lceil\tfrac{P}{D}\rceil - 1)\right] \cdot t_{rep} + t_{comp} + E(t_{div} + t_{latch}) + \lceil log_2(D)\rceil \cdot t_{lcom} + \lceil log_2(C)\rceil \cdot t_{gcom}$$

The parameter t_{learn} consists of the delay associated with the prototype vector updating operation in the corresponding NP ($\vec{w}'_J = \vec{I} \wedge \vec{w}_J$):

$$t_{learn} = [(4M + \lceil\tfrac{P}{D}\rceil - 1)] \cdot t_{rep}$$

In a single clock synchronous system, the processing rate can be analyzed using a clock period $t_{cycle} = \max\{\ t_{cc},\ \max\{t_{mem}, t_{NP}\},\ \max\{t_{vig}, t_{sub}\},\ (t_{div} + t_{latch}),\ t_{lcom},$ $t_{gcom}\ \}$. In the worst case, processing of an input vector $\vec{a}$ starts with complement coding, and ends with the update of the global winner's prototype vector:

$$t_{tot} = t_{recall} + t_{learn}$$

$$t_{tot} = t_{cycle} \cdot \left(\left[2 + 2M\lceil\tfrac{N}{P}\rceil + (\lceil\tfrac{P}{D}\rceil - 1) + E + \lceil log_2(D)\rceil + \lceil log_2(C)\rceil\right] + \left[4M + \lceil\tfrac{P}{D}\rceil - 1\right]\right)$$

$$t_{tot} = t_{cycle} \cdot \left[2M(\lceil\frac{N}{P}\rceil + 2) + 2\lceil\frac{P}{D}\rceil + E + \lceil log_2(D)\rceil + \lceil log_2(C)\rceil\right] \quad (5)$$

Notice that ceiling functions are used with the $\frac{N}{P}$ and $\frac{P}{D}$ ratios in Eq. 5, since these ratios are only valid as integer values. Clearly, Eqs. 4 and 5 can be very useful for the analysis of performance trade-offs in critical applications. For example, they show that, for a constant number of neurons N_{tot} and chips C, an increase in the number of dividers D decreases $\frac{P}{D}$ (and/or $\frac{N}{D}$), and consequently decreases the number of cycles needed for processing, but it also increases the area of an EM. For an input vector $\vec{a}$ of M elements, $\frac{N}{P}$ is a critical factor for the system's performance: indeed t_{tot} grows rapidly with $\frac{N}{P}$. Eq. 5 also justifies the need for internal memory in fast on-line applications. If the memory's access time also includes chip I/O delays, the system's performance is significantly degraded (since t_{mem} is associated with the factor $\frac{N}{P} \cdot 2M$ in t_{recall}). A relatively large number of neurons N_{tot} may also be emulated by cascading several EM chips, since C is a less significant factor in the system's overall speed. Preliminary results on estimated delays (see Tab. 2) led us to believe that: $t_{cycle} = \max\{t_{mem}, (t_{div} + t_{latch})\} = \max\{t_{mem}, \frac{t_D}{E}\}$, where $t_{rep} = t_{mem}$ is the largest fixed delay, and $(t_{div} + t_{latch})$ varies with E. Beyond some value $E = E_{limit} = \lceil\frac{t_D}{t_{cycle} - t_{latch}}\rceil$, t_{cycle} is minimized since it depends entirely on the fixed value t_{mem} ($t_{cycle} = t_{mem} > (t_{div} + t_{latch})$).

5.2: Application to radar ESM systems

By estimating the system's performance for a range of feasible configurations, we wish to obtain appropriate values for: N, P, D, $\frac{N}{P}$, $\frac{P}{D}$, $\frac{N}{D}$, when $M = 16$, $N_{tot} \geq 250$, $b = 11$ bits, $\beta = 1$, and the input processing speed is $t_{tot} = 2\mu$s. The simulation results also offer a better understanding of the design trade-offs associated with the implementation of the Fuzzy ART system architecture.

Tab. 2 presents preliminary area and delay measures for the system modules used within the cost function. These measures were obtained through synthesis using Synopsis design tools, with Nortel's 0.8μm BiCMOS technology [25], for a library of cells developed by the Canadian Microelectronics Corporation (CMC). Note that these measures are highly dependent on the technology used and on the degree of optimization invested in each module. Any conclusions drawn on results from the area-times estimation model must consider these factors. A nonrestoring division array [28] was implemented to compute the divisions. With this divider, we estimated the area of a level of pipelining registers to be $A_{level} = [(2B-1) + \frac{(E+1)}{2}] \cdot A_{latchbit}$. The area of a bit A_{membit} was estimated using a BiCMOS 1-Port SRAM block provided by the CMC [27], $A_{latchbit}$ was obtained from an 11 bit shift register, and t_{latch} was obtained by circuit simulation of a rising edge D flip-flop.

Since the delays and areas shown do not account for the effects of interconnects, and the only available parameters are for typical process as opposed to worst case, the final delay values used were all conservatively doubled when used for cost estimation. Areas were used directly but similar doubling should be expected. For example, during the simulations $t_{cycle} = 2 \times t_{mem} = 10ns$ is used. To achieve a minimum processing time and area (i.e. a minimum value $E \cdot A_{level}$ in Eq. 4), we set $E = E_{limit}$ and $t_{cycle} = t_{mem} = (t_{div} + t_{latch})$. Then, memory access time t_{mem} is the system's processing speed bottleneck. Fig. 8 presents examples of the cost simulation results (from Eqs. 4 and 5) that can be used to deduce an appropriate set of system parameters for $C = 4$, $P \geq D$, $N \geq P$, and $E = E_{limit} = 20$ levels. Fig. 8(a) shows the effect of varying P and D on cost values when $N = 64$. The results show that a smaller number of dividers D does not always yield a lower cost. It was also observed that minimum cost values are obtained for lower $\frac{P}{D}$ ratios as the value D grows. For instance, the minimum costs range from $\frac{P}{D} \simeq 20$ with $D = 1$, to $\frac{P}{D} \simeq 2$ with $D = 20$. This indicates that lower costs can be attained with smaller P values as D grows. Fig. 8(b) shows the effects of varying N and D on the cost values, when $D = 4$. Small N values give lower costs, and optimal $\frac{N}{P}$ ratios grow slowly with N. Specifically, the minimum costs range from $\frac{N}{P} \simeq 2$ with $N = 10$, to $\frac{N}{P} \simeq 2.5$ with $N = 100$. This result emphasizes the importance of selecting a relatively small ratio of neurons per NP, $\frac{N}{P}$.

Fig. 9 shows the contour plot of N versus P obtained from Eqs. 4 and 5, assuming the processing rate constraints $t_{cycle} = 10$ns and $t_{tot} = 2\mu$s, and the area constraint $\frac{A_{tot}}{C} = 20\text{mm}^2$. As a general rule, when t_{cycle} is reduced or t_{tot} grows, more neurons per NP and/or more NPs per divider still allow meeting area and delay requirements. The rather small area limit per chip was selected since module interconnections and pad frame are expected to double it. The region inside the contour plot of Fig. 9 corresponds to values of N, P, and D that can meet these area-time constraints. The diagonal lines shown correspond to the number of dividers D, ranging from 2 to 17. Fig. 9 shows that at least 2 dividers and 14 neural processors are required to meet the speed constraint. As for the area constraint, no more than 17 dividers or 60 neural processors can fit in a chip. Since our system design goal is to maximize the number of neurons per chip, N, or minimize the number of chips ($C = \frac{N_{tot}}{C}$), the upper portions of the region corresponds to good parameter choices. The maximum number of neurons that can be put on one chip while meeting the speed and area requirements is 93. This is obtained by choosing 4 dividers and 31 neural processors. Chip control may be complicated with this setup since not all dividers would have the same number of neural processors, and not all neural processors would emulate the same number of neurons. If P is constrained to be a multiple of D, then 92 neurons can be emulated by using 4 dividers and 32 neural processors. Moreover, if N is constrained to be a multiple of P, than 88 neurons can be emulated by using

Table 2. Delay and Area Measures Used for the Performance Estimation

Module	**Delay** (n**sec**)	**Area** (mm^2)
Fixed-point divider	t_D=75.2	A_D=0.468
Neural processor (NP)	t_{NP}=4.93	A_{NP}=0.192
RAM for prototype vectors	t_{mem}=5.0	A_{membit}=0.000337
Local and global comparator	t_{com}=3.96	A_{com}=0.0372
Pipeline register latch	t_{latch}=1.23	$A_{latchbit}$=0.000463

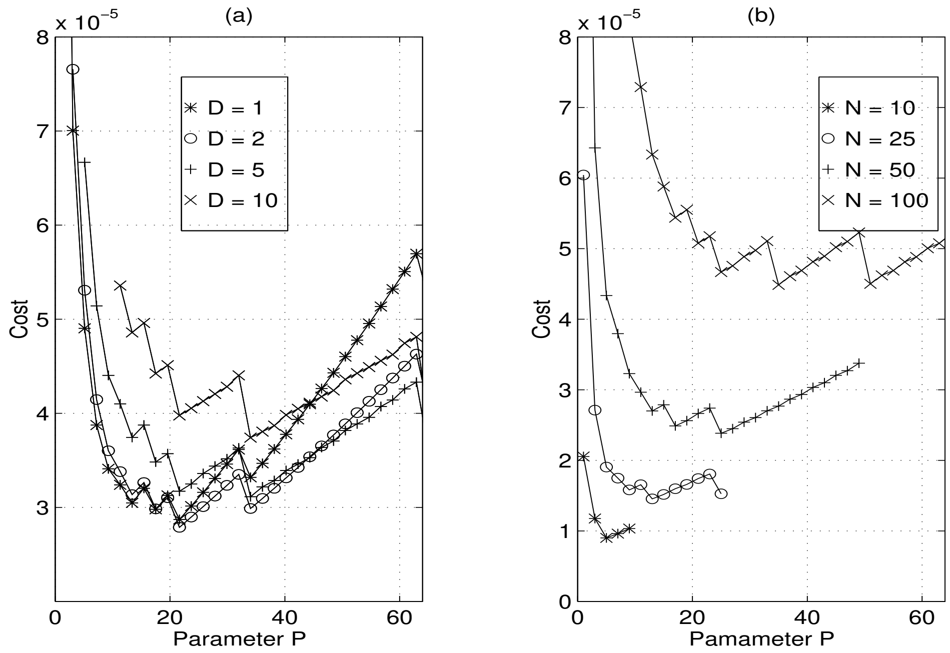

Figure 8. Cost ($A_{tot} \cdot t_{tot}$) simulation results: (a) Cost versus P for $N = 64$ (with different D values); (b) Cost versus P for $D = 4$ (with different N values).

2 dividers and 44 neural processors. Both configurations lead to a 3 chip system capable of categorizing 500000 patterns per second. A proof-of-concept, digital VLSI implementation is currently under development at cole Polytechnique de Montral. The design has been described with VHDL, and is being targeted to BiCMOS 0.8 μm technology [26], using Cadence, and the Synopsys simulation and synthesis tool.

6: Conclusion

In this paper, the Fuzzy ART neural network's algorithm has been reformulated to offer more efficient solutions for its digital VLSI hardware implementation. A dedicated system architecture that partitions this reformulated algorithm into several identical integrated circuits has then been proposed. This system architecture is suitable for high throughput clustering applications where a high degree of accuracy must be maintained. It is currently being considered for on-line clustering of intercepted radar pulses in ESM systems (i.e., data reduction). An area-time estimation model has also been presented for this system architecture. This model may help designers of rapid clustering systems in the selection of an appropriate system configuration (or system parameters), given the constraints of their application. An example of system configuration choice is demonstrated for a radar signal clustering problem. The estimation results support the general approach, and the practicality of a digital VLSI implementation.

The VLSI system architecture presented is just one among many possible architectures for the Fuzzy ART algorithm. For example, it would be possible to distribute

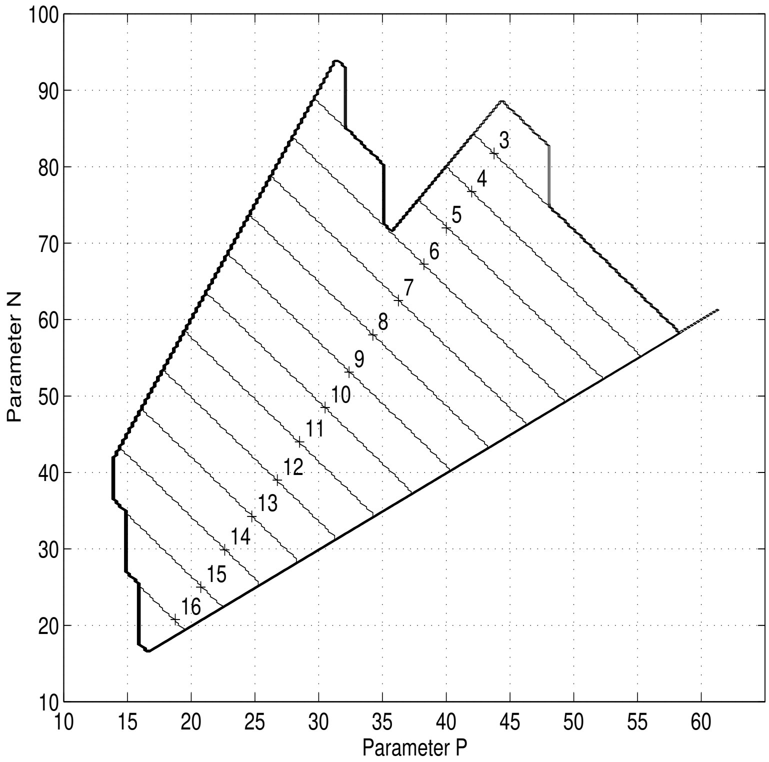

Figure 9. Contour plot of N versus P for chip configurations meeting the constraints $\frac{A_{tot}}{C} = 20\text{mm}^2$, $t_{tot} = 2\mu\text{s}$, and $t_{cycle} = 10n\text{s}$. The number of dividers is given by the diagonal lines.

bit-serial dividers in each NP, rather than having global dividers for a group of NPs. Another option would be to replace the division by another operation. It would also be useful to include a mechanism to collect statistics relative to the categories created (i.e. time elapsed since last choice, selection, update, etc.). This would permit rapid reinitialization of infrequently selected categories and reduction of the system's global processing requirements. The results of our current implementation, and future exploration should offer greater insight into the trade-offs involved in a Fuzzy ART system's implementation, and uncover an even broader range of effective architectural solutions.

References

[1] P. M. Grant and J. H. Collins, *Introduction to Electronic Warfare*, Proc. of IEE (UK), vol. 129, Part. F. no. 3, pp. 113-129, June 1982.

[2] J. F. Crespo, P. Lavoie and Y. Savaria, *Fast Convergence with Low Precision Weights in ART1 Networks*, Proc. of 1994 IEEE Int'l Symp. Circuits and Systems, London, England, vol. 6, pp. 237-204, 1994.

[3] Behrooz Kamgar-Parsi, Behzad Kamgar-Parsi and John C. Sciortino Jr., *Automatic Data Sorting Using Neural Network Techniques*, Naval Research Laboratory Report NRL/FR/5720-96-9803, February 1996.

[4] J. A. Anderson, M. T. Gately, P. A. Penz and D. R. Collins, *Radar Signal Categorization Using a Neural Network*, Proc. of the IEEE, vol. 78, no. 10, pp. 1646-1656, October 1990.

[5] M. R. Anderberg, *Cluster Analysis for Applications*, Academic Press, 1973.

[6] R. C. Dubes and A. K. Jain, *Algorithms for Clustering Data*, Prentice Hall, 1988.

[7] G. A. Carpenter, S. Grossberg and D. B. Rosen, *Fuzzy ART: Fast Stable Learning and Categorisation of Analog Patterns by an Adaptive Resonance System*, Neural Networks, vol. 4, pp. 759-771, 1991.

[8] G. A. Carpenter and S. Grossberg, *A Massively Parallel Architecture for Self-Organizing Neural Pattern Recognition Machines*, Computer Vision, Graphics, and Image Processing, vol. 37, pp. 54-115, 1987.

[9] G. A. Carpenter and S. Grossberg, Eds., *Pattern Recognition by Self-Organizing Neural Networks*, The MIT Press, 1991.

[10] B. Moore, *ART1 and Pattern Clustering*, In Proc. of 1988 Connectionist Models Summer School, Eds. Touretski, Hinton and Sejnowski, Morgan Kaufmann Publishers, pp. 174-185, 1988.

[11] S. W. Tsay, N. El-Leithy and R. W. Newcomb, *CMOS Realization of a Class of Hartline Neural Pools*, Proc. IEEE Int'l Symp. Circuits and Systems, pp. 2417-2420, New Orleans, LA, May 1990.

[12] S. W. Tsay and R. W. Newcomb, *VLSI Implementation of ART1 Memories*, IEEE Transactions on Neural Networks, vol. 2, no. 2, pp.214-221, March 1991.

[13] C. S. Ho, J. J Liou, M. Georgiopoulos and C. Christodoulou, *A Mixed Analog/Digital VLSI Design and Simulation of An Adaptive Resonance Theory (ART) Neural Network Architecture*, Simulation, vol. 66, iss. 1, pp. 31-42, 1996.

[14] T. P. Caudell, *Hybrid Optoelectronic Adaptive Resonance Theory Neural Processor, ART1*, Applied Optics, vol. 31, no. 29, pp. 6220-6229, 1992.

[15] A. Rao, M. R. Walker, L. T. Clark and L. A. Akers, *Integrated Circuit Emulation of ART1 Networks*, Proc. First IEEE Conference on Artificial Neural Networks, pp. 37-41, 1989.

[16] A. Rao, M. R. Walker, L. T. Clark, L. A. Akers and R. O. Grondin, *VLSI Implementation of Neural Classifiers*, Neural Computation, vol. 2, pp. 35-43, 1990.

[17] T. Serrano-Gotarredona, B. Linares-Barranco and J. L. Huertas, *A Real Time Clustering CMOS Neural Engine*, Advances in Neural Information Processing Systems 7, Eds., G. Tesauro, D. Touretzky, T. Leen, pp. 755-60, MIT Press, 1994.

[18] D. C. Wunsch II, T. P. Caudell, C. D. Capps, R. J. Marks II and R. A. Falk, *An Optoelectronic Implementation of the Adaptive Resonance Neural Networks*, IEEE Transactions on Neural Networks, vol. 4, no. 4, pp. 673-684, July 1993.

[19] J. S. Kane and M. J. Paquin, *POPART: Partial Optical Implementation of Adaptive Resonance Theory 2*, IEEE Transactions on Neural Networks, vol. 4, no. 4, pp. 695-702, July 1993.

[20] K. Ratanapan and C. H. Dagli, *Implementation of ART1 Architecture on CNAPS Neurocomputer*, Proc. of the SPIE - Intern. Society Optical Eng., vol. 2492, Iss. pt.1, pp. 104-10, 1995.

[21] U. Ramacher, *Guidelines to VLSI Design of Neural Nets*, In VLSI Design of Neural Networks, Eds. Ramacher and Ruckert, pp. 2-34, 1991.

[22] S. R. Jones, K. M. Sammut, C. H. Neilson and J. Strauntrup, *Toroidal Neural Networks: Architectures and Processor Granularity Issues*, In VLSI Design of Neural Networks, Eds. Ramacher and Ruckert, pp. 229-254, 1991.

[23] S. Y. Kung, *Digital Neural Networks*, Prentice-Hall, pp. 337-406, 1993.

[24] S. R. Jones, K. M. Sammut and J. Hunter, *Learning in Linear Systolic Neural Network Engines: Analysis and Implementation*, IEEE Transactions on Neural Networks, vol. 5, no.4, pp. 584-593, July 1994.

[25] R. Hadaway, P. Kempf, P. Schvan, M. Rowlandson, V. Ho, J. Kolk, B. Tait, D. Sutherland and G. Jolly, *A Sub-Micron BiCMOS Technology for Telecommunications*, Proceedings, 21st European Solid State Device Research Conference, pp. 513-516, 1991.

[26] Canadian Microelectronics Corporation, *BiCMOS Edge Support, Version 1.0*, Juin 1993.

[27] Canadian Microelectronics Corporation, *cmc128 × 32 (128 word × 32 bit) BiCMOS 1-Port SRAM*, August 1995.

[28] J. F. Cavanagh, *Digital Computer Arithmetic, Design and Implementation*, McGraw-Hill Inc., pp. 289-301, 1984.