An Associative Memory with Neural Architecture and its VLSI Implementation

Ulrich Rückert

Bauelemente der Elektrotechnik
Universität Dortmund
D-4600 Dortmund 50, F.R. Germany

Abstract

Two VLSI special-purpose hardware implementations of an associative memory model are described: a pure digital and a mixed analog/digital architecture. Both architectures can be easily extended to large scale memories with several million storage elements. The advantages and disadvantages of both architectures are pointed out. The memory concept is based on a simple matrix structure with nxm binary elements, the connections, and on distributed storage of information like artificial neural networks. There is no asynchronous feedback and the inputs and outputs are binary, too. Though the system concept is very simple, it has an asymptotic storage capacity of 0.69·m·n bits and the number of patterns that can be stored with low error probability is much larger than the number of columns (artificial neurons). The important aspect for applications is that the input and output patterns have to be sparsely coded.

1 Introduction

In the last decade there has been an increasing interest in the use of artificial neural networks in various applications. One of these application areas, where the analysis of the performance of a neural network approach is comparatively advanced is associative memory. Neural networks are well suited for the implementation of associative correlation memories [1]. The idea is that information is stored in terms of synaptic connectivities between artificial neurons. The activities of the neurons represent the stored patterns. Many different models have been discussed in literature under such names as *"Lernmatrix"*, *"Correlation Matrix"*, *"Associative Memory"* etc. [1-4]. Recent advances have been largely supported by simulations on conventional computers. However, if these models should offer a viable alternative for storing and processing information in large scale applications (e.g. pattern recognition) these systems will have to be implemented in hardware. Because of their regular and modular structure, neural networks are well adapted for VLSI system design. Implementing large numbers of individually primitive processing elements directly in VLSI hardware is intuitively appealing. There are two different approaches for supporting these models on parallel VLSI hardware [5]:

* *General-Purpose Neurocomputers* : generalized, programmable neural computers for emulating a wide range of neural network models, thus providing a framework for executing neural models in much the same way that traditional computer address the problems of "number crunching".

* *Special-Purpose VLSI-Systems* : specialized neural network hardware implementations that are dedicated to a specific neural network model and therefore have a potentially higher performance than Neurocomputers.

This paper is devoted to a special-purpose hardware implementation of a very simple associative memory loosely based on neural networks. The memory has a simple matrix structure with binary elements (connections, synapses) and performs a pattern mapping or completion of binary input/output vectors. To the authors knowledge, this comparatively simple model of a distributed associative memory was first discussed by Willshaw et. al. [3] in 1969. However, similar structures have been more generally discussed, e.g. by Kohonen [1]. The characteristics of the implemented model are described in section 2.

The important aspect for VLSI implementation of this simple memory model is the close relationship to conventional memory structures. Hence, it can be densely integrated and large scale memories with several thousands of columns (model neurons) can be realized with current technologies already. Furthermore, the regular topology results in a rigorous modularization of the system indespensible for a successful management of the design and test complexity of VLSI systems. In this respect a pure digital and a mixed analog/digital VLSI architecture are described in section 3 and discussed in section 4.

0073-1129/91/0000/0212$01.00 © 1991 IEEE

2 The Associative Memory Concept

In general the basic operation of an associative memory is a certain mapping between two finite sets X and Y. In a more abstract sense these two sets may be regarded as questions and answers or stimuli and responses, both coded as vectors of numbers (Figure 1).

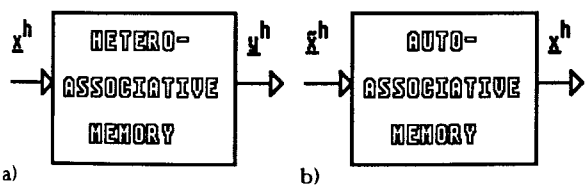


Figure 1: Basic operations of an Associatiove Memory: heteroassociation (a) and autoassociation (b).

The associative memory should respond with $\underline{y}^h$ to the input $\underline{x}^h$ for every pair $(\underline{x}^h,\underline{y}^h)$ stored in the memory. The paired associates can be selected freely, independently of each other. This operation is often called pattern mapping or heteroassociative recall [1,4]. Further it would be convenient if the associative memory responds with $\underline{y}^h$ not only to the complete input $\underline{x}^h$ but also to sufficiently large parts of it. In other words the mapping should be fault-tolerant to incomplete or noisy versions of the input pattern. A special case of this functionality is the autoassociative memory where the stored pairs look like $(\underline{x}^h,\underline{x}^h)$. Given a sufficiently large part of $\underline{x}^h$ the memory responds with the whole pattern $\underline{x}^h$ (pattern completion). Besides the discussion of the fuzzy term "sufficiently large" the input can be any part of the stored pattern and it even can be a noisy version of $\underline{x}^h$. This operation is called the best match search in terms of pattern recognition.

Among the many different implementations of an associative memory in the field of neural networks the following simplest type is very attractive as well as effective in regard to VLSI implementation. The Associative Matrix (AM) is a nxm matrix of binary storage elements w_{ij}, the connection weights. The input vectors $\underline{x}^h$ as well as the output vectors $\underline{y}^h$ take a binary form (Figure 2). The mapping is built up in the following way: The input vector $\underline{x}^h$ as well as the output vector $\underline{y}^h$ of every pair which should be stored in the AM (h=1, ... ,z) are applied to the matrix simultaneously. At the beginning all storage elements in the matrix are zero. Each storage element at the crosspoint of an activated row and column ($x_j^h = y_i^h = 1$) will be switched on, whereas all the other storage elements remain unchanged. This clipped Hebb-like rule [4] programs the connection matrix, and the information is stored in a distributed way (Figure 3 a,b):

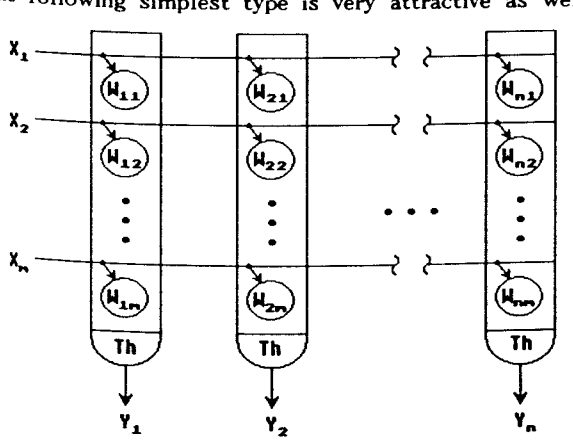


Figure 2: Structure of an Associative Matrix (AM).

$$w_{ij}^h = w_{ij}^{h-1} \vee (x_j^h \wedge y_i^h) \;,\; w_{ij}^0 = 0 \;,\; h = 1, \ldots ,z \qquad (1)$$

The recall of the constructed mapping is done by applying an input vector to the rows of the matrix. For each column i we add the products of the input components x_j and the corresponding connection weights w_{ij}:

$$S_i = \sum_{j=1}^{m} x_j^h \cdot w_{ij} \qquad (2)$$

The associated binary output vector is obtained by the following threshold operation (Figure 3c):

$$\tilde{y}_i^h = \begin{cases} 1, \text{ if } S_i \geq Th \\ 0, \text{ otherwise} \end{cases} \;,\; Th \in \mathbb{N} \text{ (threshold)} \qquad (3)$$

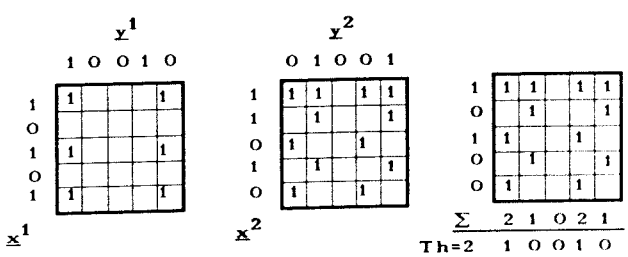


Figure 3: Programming two pattern pairs into an Associative Matrix (a,b) and pattern mapping of an incomplete input pattern (c).

Obviously, because of the above mentioned programming rule the memory matrix gets more and more filled (the connections will never be switched off). Consequently, the output might contain more '1's than the desired output pattern. The chance that this kind of error will occur increases with the number z of stored pairs. This fact causes the following quantitative questions:

i) How many patterns can be stored in an AM?
ii) How many bits of information can be stored in an AM?

Both questions were answered by Palm [6]. Summarizing his results, an AM has its optimal storage capacity I for sparsely coded input/output patterns. This means, the number l (k) of active components ('1') in the input (output) patterns should be logarithmically related to the pattern length (n,m). Asymptotically, the optimal storage capacity for heteroassociation is given by [6]:

$$I(m,n,k,l,z) \longrightarrow \ln 2 \cdot m \cdot n \qquad (4)$$

for $n,m \longrightarrow \infty$ and parameters:

$$k = \log(n) \;,\; l = \log(m) \;,\; z \leq \ln 2 \frac{m \cdot n}{k \cdot l} \qquad (5)$$

Hence, the storage capacity I is proportional to the number of storage elements n·m. Furthermore, the number of patterns z that can be stored is much larger than the number of columns (artificial neurons). For example, an optimum of I=593000 bits for n, m=1000, z=34780 (k=2,l=9) can be stored in the AM under the constraint that on the average 90% of the information of the output vector of each pair is stored [6]. Figure 4 shows the information I that can be stored in an AM as a function of the number of stored patterns (z) and Table 1 shows I as a function of the number of activated components in the input (l) and output (k) pattern respectively.

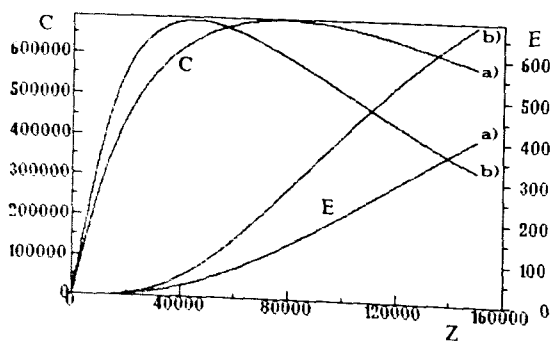


Figure 4: Storage capicity C in bits and the expected number E of additional '1's in the output pattern as a function of the number of stored patterns z (m,n = 1000):
a) k = l = 3; b) k = l = 4.

m=n	l	z	C/m·n
1024	10	23.313	0.565
2048	11	96.012	0.591
4096	12	318.353	0.604
8192	13	1.182.378	0.613
16384	14	4.407.707	0.621

Table 1: Number of patterns that can be stored with low error probability and the corresponding storage capacity C as a function of parameters n,m,l (k = 3).

For the autoassociative case the optimal storage capacity is at most half the optimal capacity for heteroassociation [7]. This is because autoassociation leads to a symmetric weight matrix and hends only half of the matrix is used for storing information.

Furthermore, it turns out that the AM works for pattern mapping applications in a more economic way compared to conventional methods (e.g. hashing) and other neural network models, if the number of patterns is large and their individual information content small [7]. These results encourage a hardware implementation in VLSI of this simple associative memory model in situations where such a mapping is a more natural way of storing information than a listing. Especially, because the AM works the more effectively the larger the matrix is [7].

3 VLSI Implementation

The Associative Matrix can be handled most flexibly as a simulation program on a conventional computer (workstation), of course. It could be shown that even serial simulation of the AM would have to perform less operations than a conventional implementation in terms of bitwise mask operations [8]. For the special case m<<n and a sparse matrix the serial implementation is even for a large number of patterns fast enough for certain applications [8]. But in general, a serial implementaton has a poor performance, especially if applied to larger matrices.

It is quite obvious that operation can be sped up considerably by parallel processing. The implementation by means of multiprocessor architectures (SIMD machines) is a promising compromise between flexible modelling - the system is still program controlled - and a complete parallel processing of large matrices. In fact, at least two research groups have already designed a parallel associative computer (SIMD) based on a set of conventional microprocessors communicating via a common bus [9,10].

Consequently, the highest degree of parallelism is achieved by task-dedicated VLSI systems. It is well in the range of current technologies to implement an AM effectively on VLSI chips. Two different special-purpose VLSI architectures have been designed for an AM so far at the University of Dortmund: a digital and a digital/analog implementation. Both of them will be discussed in this paper.

The system architecture in both cases is split up vertically into "slices"; each slice manages an equal number of columns. The slices are controlled by a conventional microprocessor (system control, Figure 5), distributing input data in an appropriate way to the slices and collecting output data from the slices. In consequence of the sparsely coded input/output patterns the microprocessor transfers and collects the

patterns optimally by means of the addresses of the activated components. Hence, a transfer operation of a m-bit pattern takes only log(m) cycles and address lines in the serial case.

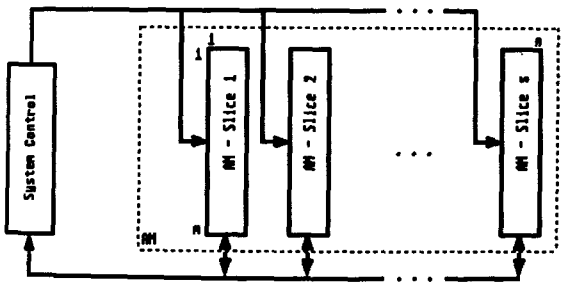


Figure 5: Partition of a mxn Associative Matrix into slices.

3.1 Digital Implementation

In the case of the digital implementation the columns of an AM are controlled by a special slice chip comprising several very simple processing units (PUs). Each PU controls one column of the matrix and computes bit-serial the weighted sum (Equation 2) of the input pattern and the respective column. Because the input/output signals as well as the connection elements are binary, the basic building blocks of a PU are a counter and a comparator (Figure 6a). The programming algorithm (Equation 1) for the connection matrix is realized by a simple OR-logic-block and is incorporated on the chip, too. The connection matrix can be built up by conventional RAMs (Figure 6b).

In order to transfer the output pattern to the system control the addresses of the '1's in the pattern are generated locally in the slice chips. All slice chips are connected to a common bus and the access to the bus can be controlled by daisy-chaining or by an additional priority encoder logic. In case of the daisy-chaining method the time for transferring the output pattern is proportional to the number of slice chips. In the other case the time is proportional to the number of active components ('1'). In both cases the transfer of the associated output pattern and the calculation of the weighted sum of inputs can be pipelined.

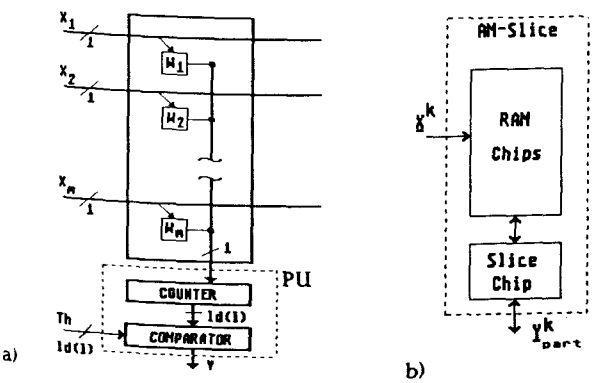


Figure 6: Basic building blocks of a digital implementation of an Associative Matrix column (a) and slice (b).

Up to now several standard-cell-designs comprising 32 PUs (e.g. 2µm CMOS, 31mm^2, ≈20000 transistors, 53 pads, 10MHz [11]) and a full-custom-design comprising 128 PUs (2µm CMOS, 64mm^2, ≈50000 transistors; Figure 7) of a slice chip have been finished. Instead of realizing a whole chip in silicon we have first fabricated and tested successfully a single PU of the full-custom-design at the University of Dortmund. The tested 6-Bit-PU is able to perform the calculations for the Equations (1) - (3) at least at a clock rate of 12 MHz. Optimization in respect to speed hasn't been done yet. Instead, we have concentrated ourselves on a modular and testable design. Therefore, the PU is built up by six bit-slices which can be configured to a scan path for test reasons. An extension to a 8- or 16-Bit-PU is easy achieveable to duplicating the bit-slices.

Based on these facts, a 8192x8192-AM can be built up by a 64 MBit-DRAM and 64 slice chips each comprising 128 PUs, for example. This AM stores more than one million sparsely coded patterns with E < 1 (Table 1) , which corresponds to the storage capacity of 40 Mbits.

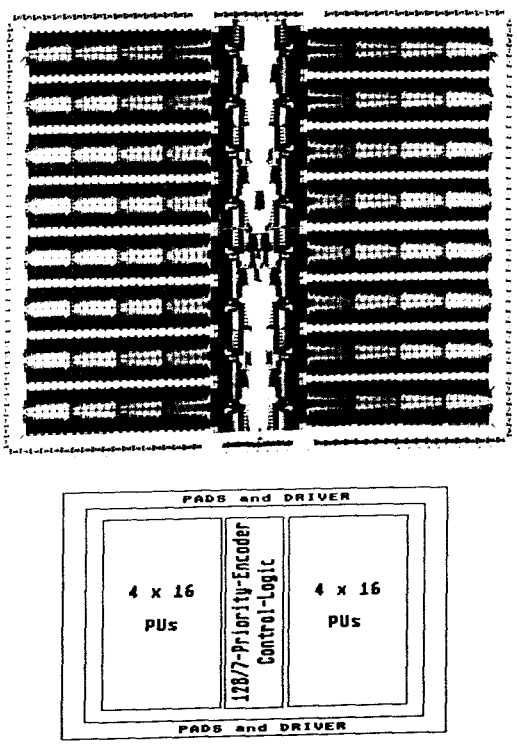


Figure 7: Layout and floorplan of a slice chip comprising 128 processing units (2 µm CMOS, one metal and one poly layer).

With current submicron technologies it is possible to integrate 512 and more PUs on a single slice chip. Hence, the limiting factor for the number of PUs per chip is the pin count. Nevertheless, it is possible to have 256 PUs on a slice chip at least. The important disadvantage of this digital approach up to now is that most RAM chips have a one, four or eight bit organization whereas a longer word length (≥16) is more appropriate for this approach. In order to get the highest degree of parallelism the optimal memory organization is m x u (m = number of matrix rows, u = number of PUs per slice chip). For the above mentioned 8kx8k-AM RAM chips with a 8kx128 organization are required, for example. Therefore, the system architecture has to be slightly modified in order to make effective use of currently available memory chips. Work on this topic is done at the moment.

Such an implementation performs a pattern mapping within 100μs. The association time is proportional to the number of '1's in the input/output patterns (log(m) + log(n)) and hence independent of the number z of stored pairs. Table 2 shows some time estimations of the association time for a single pattern (t_A) and the programming time for z pattern pairs (t_P) depending on the number of '1's in the input and output patterns. The estimations are based on test results of the fabricated PU in combination with a static RAM (100 ns cycle time).

m=n	l	k	z	t_A(μs)	t_P(s)
4096	12	3	318353	4.4	2.7
	11	10	95395	5.5	0.7
8192	13	3	1182378	4.7	11
	13	10	354750	6.1	3.3
16384	14	3	4407707	5.0	44.1
	14	10	1322288	6.4	13.2

Table 2: Association (t_A) and programming (t_P) time estimations as a function of the parameters n,m,k,l,z.

3.2 Digital / Analog Implementation

The largest computational load implementing an AM is incurred by the weighted sum of input signals (Equation 2). Using analog circuit techniques [12] , this sum can be effectively computed by summing analog currents or charge packets, for example. In Figure 8 a simple circuit concept is proposed in CMOS technology. The matrix operation is calculated by current summing and the threshold operation is done by an analog voltage comparator.

The accuracy of analog circuits is not as high as for digital circuits, but they can be built much more compactly and they are more appropriate for the

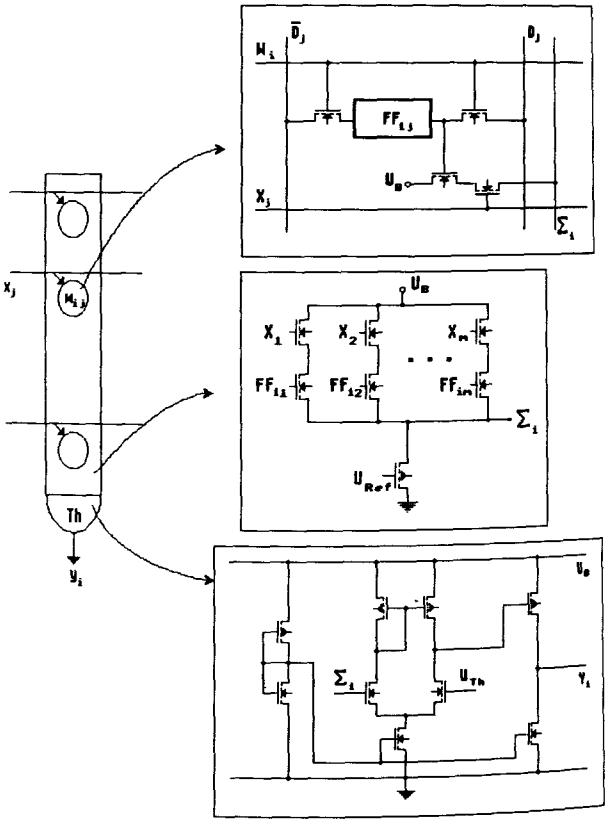


Figure 8: Analog circuit implementation of an Associative Matrix column.

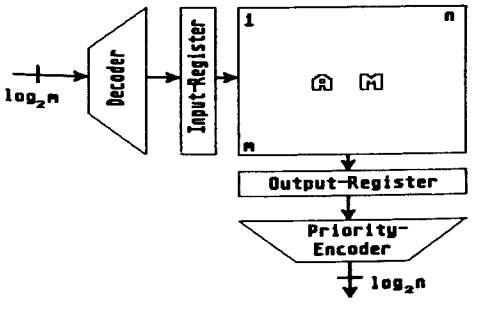
highly parallel signal transfer operations immanent in neural networks. Note, however, that analog circuits are not so densely integrated as it may seem at first glance. They demand large-area transistors to assure an acceptable precision and to provide good matching of functional transistor pairs, as used in current mirrors or differential stages. Furthermore, analog circuits are influenced by device mismatches from the fabrication process and it is very difficult to control offset voltages, for example. Consequently, analog implementations should be applied to artificial neural networks requiring only modest precision. One example for such a network is the AM because there are only log(m) terms contributing to the weighted sum S_i. The required accuracy of an AM is only about 4 to 5 bits even for large matrices (m,n>10000) and hence in the range of analog circuit techniques.

The design of the connection element is based on conventional storage devices (ROM, RAM, EEPROM). For example, a conventional static memory cell has to be enlarged by two transistors, an EEPROM cell requires no additional transistors [13]. Hence, one million programmable connections can be integrated on one chip with current VLSI-techniques. The 8192x8192-AM requires 64 of such chips each comprising 128 columns.

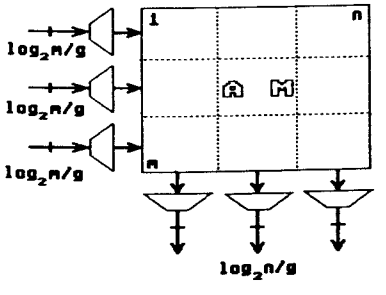
Even more limiting to the overall size of an AM slice than the area needed for the connections are the pin requirements of each slice. Taking advantage of the sparsely coded patterns permits an effectively serial as well as parallel transfer of the input/output patterns. The input/output organization in the serial case is similar to that of the digital implementation (Figure 9a). For a full parallel transfer of the patterns the m rows and n columns of the matrix are divided into g blocks of equal size. Under the assumption that at uppermost one component in each block is active, only m/g (1 of m/g)-decoders are needed for a full parallel transfer of the input pattern to the AM (Figure 9b). We calculate the number of pins as:

$$p = \log_2(m/g) \cdot g \qquad (6)$$

For $g \approx \log(m)$, a 8 192 x 128-AM-slice requires less than 130 pins. Because of the full parallel operation a recall occurs within 1 µs. Two test chips in 2.5 µm CMOS technology, a 64 x 64 AM according to Figure 9a (7 x 8 mm^2 , ≈ 40.000 transistors, Figure 10) and a 96 x 16-AM-slice according to Figure 9b (7 x 5 mm^2, ≈ 20.000 transistors [12]) with programmable connections (Figure 8) have been fabricated at the University of Dortmund. With both test chips the above mentioned functionality has been tested and verified. The accuracy of the proposed analog circuits turned out to be at least 4 bit. In other words, sparsely coded binary patterns with up to 16 activated components can be handled correctly by this simple implementation.



a)



b)

Figure 9: Input/output organization of a serial (a) and full parallel (b) analog implementation of an Associative Matrix.

Because of the modular and regular structure of the architecture, the implementation of large AMs (n,m > 10.000) is feasible. A further attractive feature of the AM is its fault tolerance to defective connections. Even in the presence of 5% defects, up to 20.000 sparsely coded patterns (l=13, k=3) can be stored in a 1.000 x 10 00 AM with low error probability. Therefore, the AM will also be well adapted for the evolving wafer-scale-integration technique.

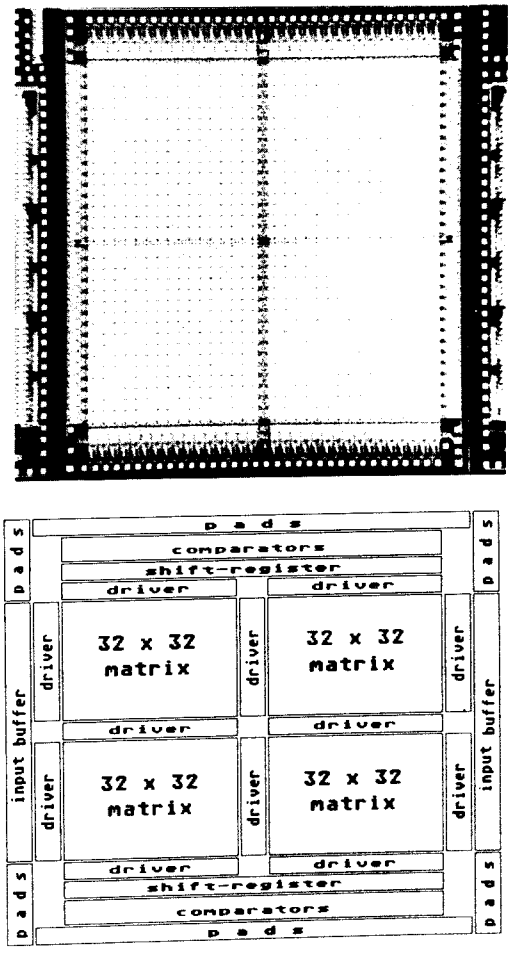


Figure 10: Microphotograph and floorplan of the analog/digital implementation of the 64x64 Associative Matrix test chip.

4 Discussion

Though the AM system concept is comparatively simple, it has very attractive features in regard to other associative or neural network VLSI implementations:

* the asymptotic storage capacity is 0.69·n·m bits
* the number of sparsely coded patterns that can be stored in an AM is much larger than the number of columns (artificial neurons)
* the number of operations during association is only O(log(n)·m) instead of O(m·n)
* the simpler circuit design requires less silicon area

Because of the modular and regular structure of the proposed architectures, the implementation of very large AMs (n,m≥10.000) is feasible. This aspect is very important for practical applications where the AM has to be extended to a useful number of storage elements. Work on possible applications of an associative memory of this type is done at the moment by different research groups, e.g. in the field of speech recognition, scene analysis and information retrieval.

Comparing both VLSI approaches presented above, we can call on efficient software tools for a fast, reliable and even complex digital system design. For the memory matrix we can use standard RAM chips employing the highest density in devices. In general, the matrix dimensions (n,m) can be extended by using additional RAM chips. An important disadvantage up to now is that most RAM chips have a one, four or eigth bit organization whereas a longer word length (≥16) is more appropriate for the digital implementation.

On the contrary, the design of analog circuits demands much more time, good theoretical knowledge about transistor physics and a heuristic experience of layout. Only a few process lines are characterized by analog circuits. The noise immunity and precision is low compared to digital circuits. The fixed matrix dimensions are a further disadvantage. In their favor, we point out that analog circuits can be built much more compactly and are more appropriate for the highly parallel signal transfer operations immanent in neural networks. For example, a 1.000x1.000 AM can be integrated on one chip, whereas the digital concept requires several slice and RAM chips. In conclusion, both approaches have their advantages and it remains to be seen which type of implementation will be more effective in certain applications.

Acknowledgements

The author thanks the DFG (Deutsche Forschungsgemeinschaft) for financial support and Thomas Will for many fruitfull discussions.

References

[1] T. Kohonen: "Associative Memory: A system-theoretical approach", Springer Verlag, Berlin 1977.

[2] K. Steinbuch: "Die Lernmatrix", Kybernetik 1, Heft 1, pp. 36-45, 1961.

[3] D.J. Willshaw, O.P. Buneman, H.C. Longuet-Higgins: "A Non-Holographic Model of Associative Memory", Nature 222, no. 5197, pp. 960-962, 1969.

[4] G. Palm: "Neural Assemblies", Springer Verlag, Berlin 1982.

[5] P.C. Treleaven: "Neurocomputers", Int. Journ. of Neurocomputing, Vol.1 89/1, pp. 4-31, 1989.

[6] G. Palm, "On Associative Memories", in Physics of Cognitive Processes, E.R. Caianiello (ed.), World Science, pp. 380-420, 1986.

[7] G. Palm, "On Associative Memory", Biol. Cybern. 36, pp. 19-31, 1980.

[8] H.J. Bentz, M. Hagstroem, G. Palm: "Information Storage and Effective Data Retrieval in Sparse Matrices", Neural Networks, Vol. 2, No. 4, pp. 289-293, 1989.

[9] G. Palm, T. Bonhoeffer: "Parallel Processing for Associative and Neural Networks", Biol. Cybern. 51, pp. 201-204, 1984.

[10] U. Rückert, J. Moschner: "A SIMD Architecture for Parallel Simulation of Associative Networks, to be published, 1990.

[11] U. Rückert : "VLSI Implementation of an Associative Memory based on distributed Storage of Information", in: Neural Networks, ed. L.B. Almeida, C.J. Wellekens, Lecture Notes in Computer Science No. 412, Springer Verlag, pp. 267-276, 1990.

[12] K. Goser, U. Hilleringmann, U. Rückert, K. Schumacher: " VLSI Technologies for Artificial Neural Networks", IEEE Micro, Dec. 1989, pp. 28-44, 1989.

[13] U. Rückert, I. Kreuzer, K. Goser : " A VLSI Concept for an Adaptive Associative Matrix based on Neural Networks", COMPEURO, Hamburg May 1987, Proceedings pp. 31-34, 1987.