ULSI Architectures for Artificial Neural Networks

NO CLEAR CONSENSUS EXISTS ABOUT HOW TO EXPLOIT THE POTENTIAL FOR MASSIVELY PARALLEL IMPLEMENTATIONS OF ARTIFICIAL NEURAL NETWORKS. THREE HARDWARE IMPLEMENTATIONS DEMONSTRATE KEY ISSUES SURROUNDING THEIR USE.

Ulrich Rückert
University of Paderborn

The ongoing revolutionary progress of microelectronics is the driving force behind the constant development of new technical products that have markedly improved functionality and higher performance, yet at a lower cost. We expect this trend to continue beyond the year 2015. The challenge lies in mastering the resulting design complexity and achieving economic viability for integrated systems with more than 100 million devices per square centimeter. This requires system concepts that both exhaust the possibilities of semiconductor technology and reduce the design and test complexity. Because of their highly regular, modular structure, information processing parallelism, inherent fault tolerance, learning ability, and environmental adaptability, artificial neural networks (ANNs) offer an attractive alternate approach for ultra-large-scale-integration (ULSI) systems. ANNs offer a variety of techniques for use in resource-efficient information processing architectures, and, in particular, for cognitive systems. The three approaches we examined are model-specific integrated circuits for neural associative memories, self-organizing feature maps, and local cluster neural networks.

With structure sizes smaller than 0.1 micron, semiconductor technology starts falling below the level of biological structures forming the brain. However, the brain efficiently uses all three dimensions, whereas microelectronics can use only the two physical dimensions of the silicon die surface. Nevertheless, taking an area of one square millimeter—roughly the square dimension of a Purkinje cell (a type of neuron) in the cerebellar cortex, shown in Figure 1a—we can use 0.18-micron CMOS technology to implement a digital artificial neuron (Figure 1b) with 170,000 8-bit weights (synapses) and an 8-bit microprocessor as a neural processing unit (Figure 1c). Weights are the practical implementation (in hardware, software, theory) of (biological) synapses (contacts between nerve cells).

Two approaches exist for supporting ANNs in parallel computing architectures: general-purpose neurocomputers for emulating a wide range of neural network models, and special-purpose ULSI systems dedicated to a specific neural network model. General-purpose neurocomputers offer a high degree of observability of the inner workings of neural algorithms as well as flexibility. Special-purpose ULSI designs offer resource-efficient speed, size, and power consumption. Progress continues in both approaches, and researchers have realized many different architectures in working hardware. There exists a variety of architectures within these two approaches. This is necessary to address the difficulty in determining the best way to perform ANN

0272-1732/02/$17.00 © 2002 IEEE

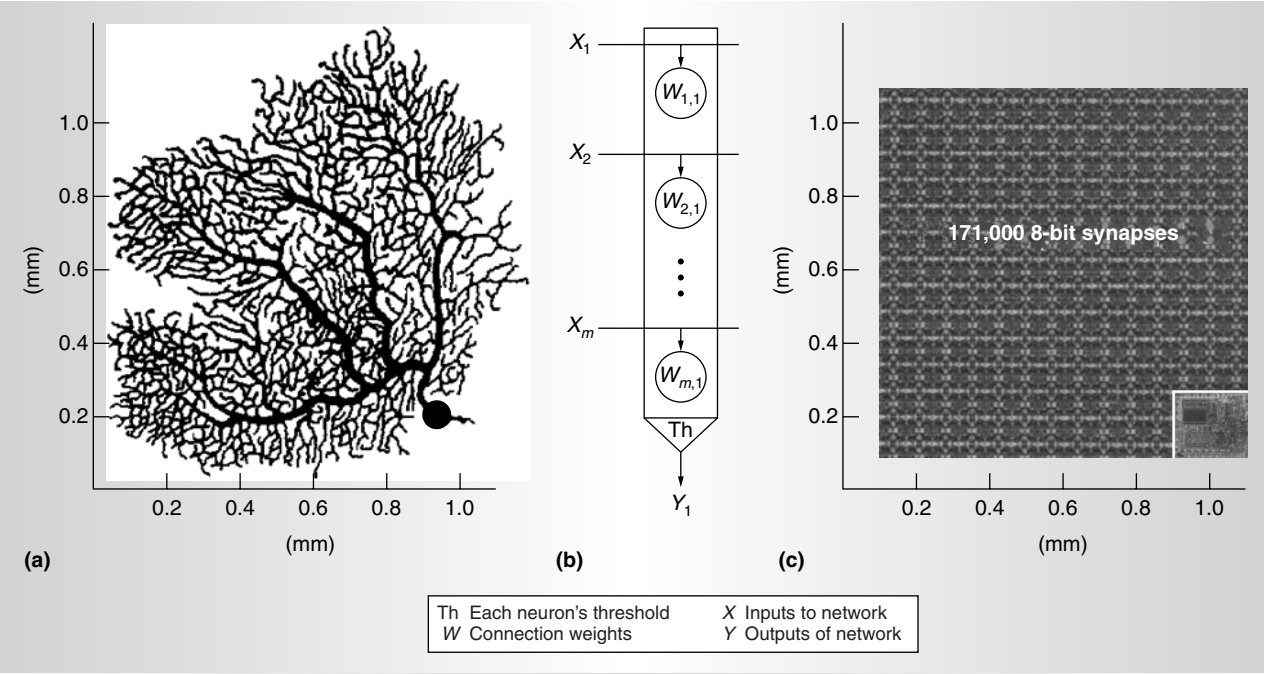


Figure 1. Principal structure of a biological neuron (a), an artificial model neuron (b), and a microelectronic implementation of an artificial neuron (c).

calculations for any given application.[1-3] Additionally, no one has given the problem of benchmarking a full treatment or found a commonly accepted adequate metric for performance evaluation.

We developed our own approach to special-purpose ULSI systems for neural networks. The basic procedure for microelectronic implementation of specific ANN models is simple:

1. Select an artificial neuron model.
2. Implement this model by means of an adapted (analog or digital) processing unit together with the required memory for the weights (synapses).
3. Make as many copies of this neuron implementation as necessary, and connect these neurons to the selected network structure.

Table 1 shows that in respect to functionality, we distinguish neural networks by association, classification, and controlling or approximation. For each functionality, we selected a neural network model well adapted to microelectronic implementation because of modular, regular architecture, required calculation accuracy, scalability, and performance. All chips were fabricated using the Europractice Services, which are offered to European universities at reasonable costs.

Neural associative memory

The basic operation of an associative memory is to map between finite pattern sets X and Y. The associative memory responds with output vector $\boldsymbol{y}^t$ to the input vector $\boldsymbol{x}^t$ for every pair (t = 1, … , z) stored in the associative memory. This mode of operation is called pat-

Table 1. Characteristic application fields for artificial neural networks.

X→ANN→Y		Task	Evaluation	Model
Discrete	Discrete	Association/memory	Storage capacity	Binary neural associative memory
Continuous	Discrete	Classification/quantization	Error probability	*n*-bit input, self-organizing map
Continuous	Continuous	Approximation/interpolation	Distance measure	Local cluster neural network

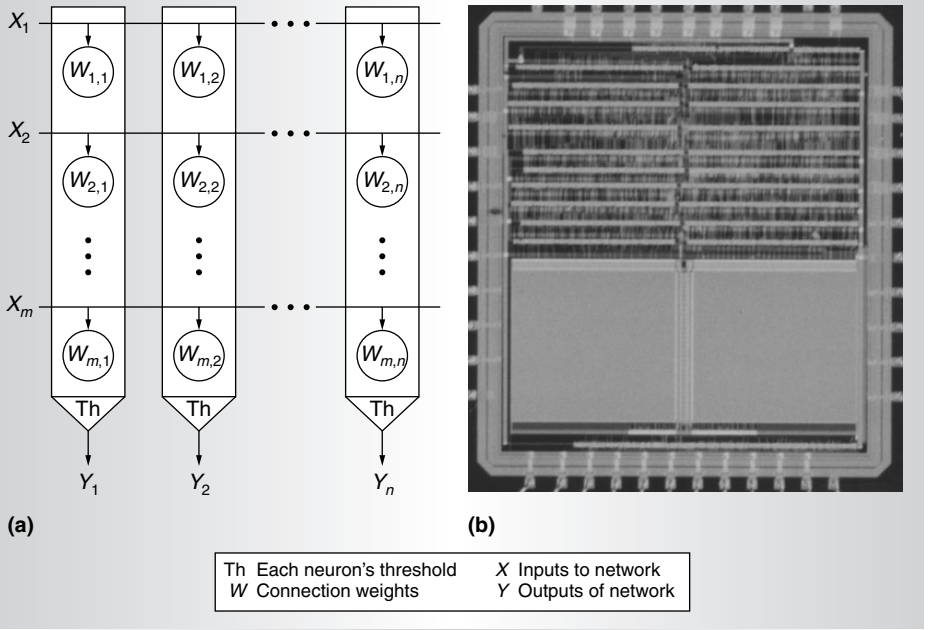


Figure 2. Architecture (a) and photograph (b) of the Saram+ chip.

tern mapping or hetero-associative recall. A special case of this functionality is the autoassociative recall (pattern completion) where the stored pairs look like ($\boldsymbol{x}^t$, $\boldsymbol{x}^t$). Given a sufficiently large part of $\boldsymbol{x}^t$, the memory responds with the whole pattern $\boldsymbol{x}^t$ (t is an index for the stored pattern pairs [t = 1, … , z]). Palm and others studied a very simple model of a neural network performing this task efficiently.[4] We call this model the binary neural associative memory (Binam) because the input, output, and connection weights (synapses) are binary.

We designed ULSI architectures for the Binam using digital, digital/analog, and analog circuit techniques.[5-7] For our implementation using serial associative RAM (Saram), we integrated the memory and the processing unit on one chip. We tested two digital implementations, which we will call Saram& and Saram+, and one analog implementation, Saram@. Both digital architectures are based on a 16-Kbit on-chip static RAM; a neural processing unit; a coding block, including input/output logic; and an on-chip controller providing 12 instructions for synchronizing, controlling, and testing the modules.

Saram& uses a fixed threshold that is set by default to the number of 1s (h bits)—the set m without h of these bits is 0—in the input pattern. The calculation unit of each neuron has about 10 standard cells or 120 transistors. The Saram+ chip, shown in Figure 2a and b, uses a programmable threshold to widen its applicability, but the neurons are more complex due to the implementation of a counter instead of a few simple logic gates. Consequently, the calculation unit of each Saram+ neuron requires 38 standard cells or 432 transistors. For both digital chips, an association of a binary input pattern having h of its m bits set to 1 (the other bits are 0) with a binary output pattern having k of its n bits set to 1 takes $2h + k + 2$ clock cycles. The learning/programming of such an association takes $3h + k$ clock cycles. Applying the maximum clock frequency we designed and tested, 25 MHz, the digital chips (Saram&, Saram+) have a typical association rate (associated patterns per second) (h = 6, k = 2) of 1.56×10^6 patterns per second or 600 million connections per second. The typical learning/programming rate in our tests is 1.25×10^6 patterns per second or 480 million connection updates per second. The average power consumption is 140 mW for Saram& and 170 mW for Saram+ for a 5-V power supply and a 25-MHz clock rate. These chips can store about 1,200 associations (sparsely coded: h = 6, k = 2) reliably. See the "Binary neural associative memory" sidebar for a detailed explanation of Binam operations.

We implemented the connection matrix (synapses) of the Binam model using a standard static memory macro. The matrix (and the macro) has a 256-word × 64-bit complexity. Hence, every input pattern is 256 bits and every output pattern is 64 bits wide. Since every weight of the connection matrix is binary, a standard six-transistor cell stores one weight. If we use dynamic memory cells, only two devices are necessary for each connection weight. Both digital chips have nearly the same die size of 25 sq. mm (made with standard cell, 1.0-micron CMOS technology), with a 64-neuron × 256-synapse matrix size. Figure 2b shows that the ratio between the memory area

Binary neural associative memory

The binary neural associative memory is a single layer feed-forward neural network with n neurons having m inputs each. The input vectors $\boldsymbol{x}^t$, the output vectors $\boldsymbol{y}^t$, and the weights w_{ij} take a binary form ($t = 1, \ldots, z$; $i = 1, \ldots, m$; and $j = 1, \ldots, n$). The associative mapping is built up in the following way: The input vector $\boldsymbol{x}^t$ and the corresponding output vector y^t of every pair stored in the binary neural associative memory are applied to the matrix simultaneously. Initially, all connection weights w_{ij} are zero. Each weight w_{ij} at the intersection of an activated row and column ($\boldsymbol{x}^t_i = \boldsymbol{y}^t_j = 1$) will be switched on, whereas all the other connection weights remain unchanged. This clipped Hebb-like[1] rule programs the connection matrix and stores the information in a distributed way. Applying an input vector to the neurons recalls the constructed mapping. For each neuron we sum up the products of the input components $\boldsymbol{x}_j$ and the corresponding connection weights w_{ij}:

$$S_j = \sum_{j=1}^{m} x_j \times w_{ij}$$

The associated binary output vector $\boldsymbol{y}$ is obtained by the following threshold operation:

$$y_j = \begin{cases} 1, & \text{if } S_j > Th \in N \\ 0, & \text{otherwise} \end{cases}$$

For an efficient application of the Binam, the input and output patterns must be sparsely coded. This means only a few [$h = \log(m)$, $k = \log(n)$] input/output lines can be active (1) at any time. This simple system concept has very attractive features for neural associative memories and content addressable techniques:

- the asymptotic storage capacity is 0.69 $n \times m$ bits,
- the number of patterns that can be stored is approximately (0.69 $n \times m$) / ($h \times k$) and hence much larger than the number of neurons n,
- the number of operations during association is O [log (m) $\times n$] instead of $O(n \times m)$, and
- the binary neural associative memory is well suited for ULSI implementation.

The application of Binams is attractive whenever requirements include a fast response and fault-tolerant access to stored patterns in large databases. Hence, the typical application area is information retrieval. Binams have efficient storage capacity, access time, and power consumption as long as special hardware is available. In addition, the Binam works more efficiently as the size of the matrix increases.

References

1. D.O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, Wiley, New York, 1949.

(weights) and the standard cell area (calculation units) for these test chips is nearly one. Increasing the width of the input patterns to $m > 8192$ (where m is the dimension of the input vector) would give a ratio of less than four percent relative usage for the neural processing unit. Furthermore, using full custom designs for the neural processing units reduces the circuitry area by about 50 percent.

We used mixed-mode analog and digital circuits for the Saram@ hardware implementation.[8] We used analog circuit techniques to compute the weighted sum within the neuron by current summing. For weight storage, we modified a six-transistor static memory cell with respect to transistor geometries. The analog concept enhances the speed of the association. In contrast, we achieved a lower accuracy (than in the digital implementation) for summing up activated 1s in the matrix, but the required accuracy for the analog circuitry grows only with the number of active vector components h = log(m) and not with the vector dimensionality itself. The Saram@ test chip has a 16 neuron × 16 synapse matrix at a 10-sq.-mm die size (using 1.2-micron CMOS technology). We tested the chip up to a 1-MHz frequency. During an association, the chip draws a 1.4-mA current, hence power dissipation for a 5-V supply voltage is about 7 mW.

The Saram chips can be scaled up on a chip as well as cascaded with other chips to build up larger Binams. For example, using 0.18-micron CMOS technology, we can implement on a chip, 4,000 neurons, each having 16,000 inputs. Cascading four such chips results in 16 K × 16 K Binams in which several million patterns are associatively stored and each pattern retrieved in about 10 ns.

Self-organizing maps

For classification tasks, we use self-organizing maps (SOM) as proposed by Kohonen[9] for microelectronic implementation. SOMs use an unsupervised learning algorithm to form a nonlinear mapping from a given high-dimensional input space to a lower-dimensional map of neurons; see the "Self-organizing maps" sidebar (next page) for a detailed description. To ease the efficient

Self-organizing maps

In general, a self-organizing map is a single layer of two-dimensionally arranged artificial neurons, as Figure A1 shows. Each neuron receives the same input vector $\mathbf{x} = (x^1, \ldots, x^n)$. First, the map has to learn a set of input vectors (learning phase). The unsupervised learning process starts with a suitable initialization of the connection weights of the neurons. If no information about the input data is available, the weights are initialized with random values. In several training epochs, all vectors of a training data set are trained to the map. The training result is a nonlinear mapping from the given high-dimensional input space to the two-dimensional map of neurons. An example of a training result is shown in Figure A2. In this case, six clusters of a four-dimensional input space are mapped to a SOM with 40×40 neurons. The neighborhood of the clusters in the input space is preserved on the map and cluster with a higher density of data points, taking a larger area on the map.

For training a vector $\mathbf{x}$ the neuron that has the most similar weight vector $\mathbf{m}$ is determined:

$$\|\mathbf{x} - \mathbf{m}_{bm}\| = \min_i \|\mathbf{x} - \mathbf{m}_i\|$$

The similarity between the input vector $\mathbf{x}$ and the so-called best-match neuron $\mathbf{m}_{bm}$ is often calculated based on the Euclidean distance. If n is the dimension of the input and the weight vectors, the Euclidean distance is defined as:

$$d = \|\mathbf{x} - \mathbf{m}\|_2 = \sqrt{\sum_{j=1}^{n} \left(x_j - \mathbf{m}_j\right)^2}$$

The best-match (or winning) neuron is adapted to the input vector by applying the following learning rule:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}\left[\mathbf{x}(t) - \mathbf{m}_i(t)\right]$$

This learning rule not only alters the winning neuron, but also neurons in the neighborhood on the map. Therefore a neighborhood function h_{ci} is defined:

$$h_{ci} = \alpha(t) \times e^{-\left(\frac{\|\mathbf{r}_{bm} - \mathbf{r}_i\|^2}{2\sigma^2(t)}\right)}$$

The term $|\mathbf{r}_{bm} - \mathbf{r}_i|$ estimates the topological distance between the best-match neuron and neuron i on the map. The position of the best-match neuron is $\mathbf{r}_{bm} \in \Re^2$, and the position of the neuron i on the planar map is $\mathbf{r}_i \in \Re^2$. The function $\sigma^2(t)$ defines the width of the Gaussian curve and $\alpha(t)$ the adaptation strength. Both are decreasing functions of the time-discrete index t (training steps). After learning, the map recalls for any given input vector the best-match neuron. In other words, the map performs a classification of known as well as unknown input data.

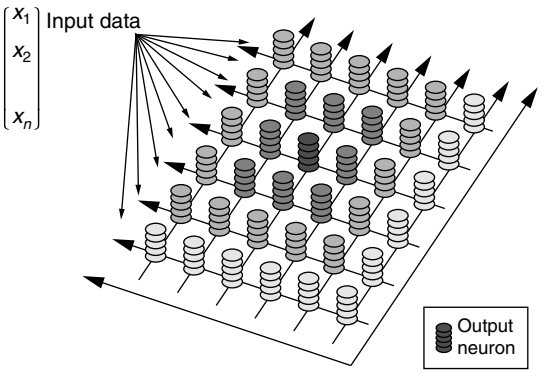
The NBISOM chips simplify the original algorithm proposed by Kohonen[1] to minimize the necessary chip area and maximize the number of processing elements per chip. Therefore, the Manhattan distance replaces the Euclidean distance to get rid of the multiplications:

$$d = \|\mathbf{x} - \mathbf{m}\|_1 = \sum_{j=1}^{\ell} \left|x_j - \mathbf{m}_j\right|$$

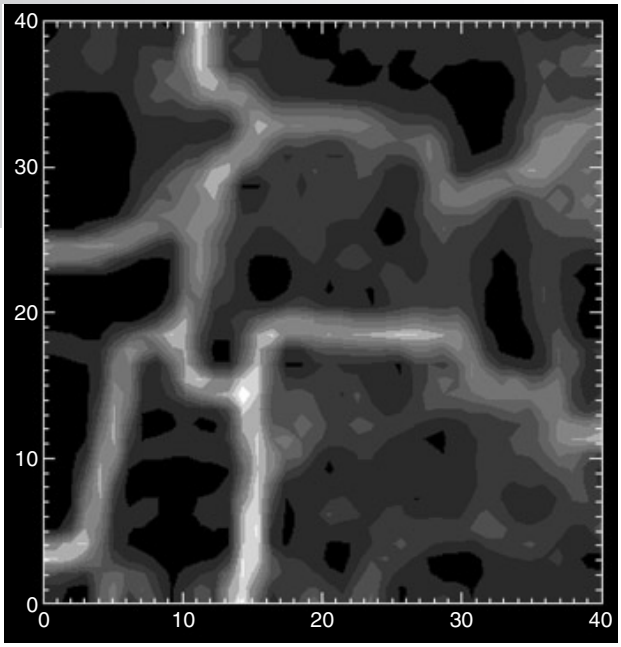
The second simplification concerns the neighborhood function. This function is restricted to a set of factors that can be realized by a simple shift-register: $h_{ci} \in \{1, 1/2, 1/4, 1/8, 1/16, \ldots\}$.

Reference

1. T. Kohonen, *Self-Organizing Maps,* Springer, Berlin, 1995.



(1)



(2)

Figure A. Self-organizing maps: architecture (1), visualization of a learning result (2).

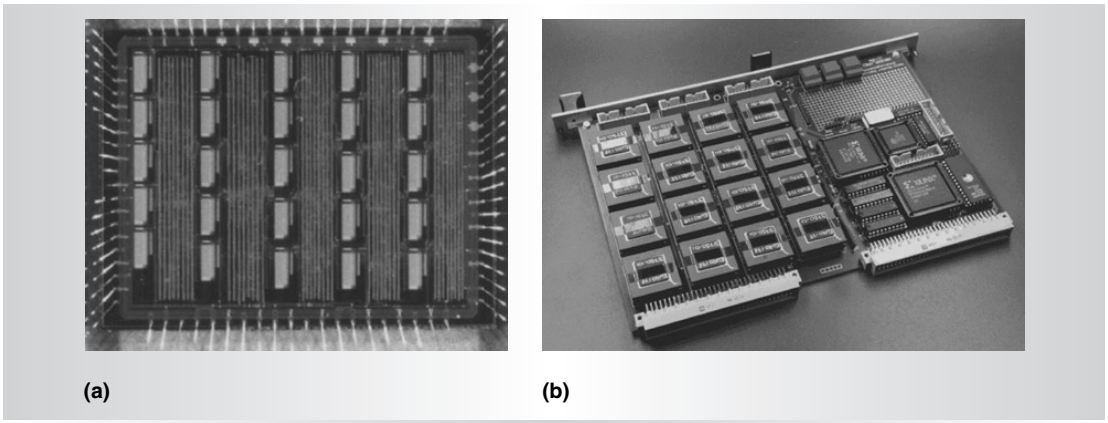


Figure 3. Self-organizing maps: photograph of NBISOM_25 chip (a), and a VMEbus-board with 16 NBISOM_25 chips (b).

implementation of SOMs in parallel hardware, we simplified the original algorithm to create the *n*-bit input SOM (NBISOM) architecture.[10] In particular, we used the Manhattan distance to calculate the distance between the input vector and the model vectors to avoid the square root multiplication and calculation required for the Euclidean distance (which is typically used in SOM implementations). Restricting the values of the adaptation factor replaces the multiplications required for adaptation with shift operations. Additionally, we set the accuracy of the input vector components and model vectors to 8 bits. We have run a large number of simulations to prove the quality of the simplified algorithm.

The given simplifications lead to a massively parallel hardware implementation, offering one processing element with internal memory for every neuron of the SOM. Thus, every processing element must have the capability to perform the calculations required for learning and recall. We developed an application-specific integrated circuit, NBISOM_25, which integrates 25 processing elements arranged in a 5×5 array, as shown in Figure 3a. Apart from the required calculation units and some interface logic, every chip contains a RAM block capable of storing 64 vector components. We implemented a controller for the interpretation of eight different commands and for communication between the processing elements themselves and with an external controller. The chip is fabricated using a 1.0-micron CMOS process with a die size of 74.3 sq. mm.

We built a board with 16 NBISOM_25 chips (Figure 3b) that integrates a self-organizing map in our VMEbus system. Thus, we can realize map sizes up to 20×20 elements with 64 vector components. For interfacing with the VMEbus and for controlling the array of processing elements, we used two field-programmable gate arrays (FPGAs). We store the input and output data in a dual-port RAM to decouple data transfer via the VMEbus from the onboard communication. Figure 3b is a photograph of the VMEbus board containing 16 NBISOM_25 chips. The maximum performance of the NBISOM_25 chips is 4,096 million connections per second during recall and 2,382 million connection updates per second during learning for a 20×20 matrix.[11] The classification rate is 160,000 patterns per second.

To improve the functionality of the NBISOM_25 chips we developed the NBX chip.[12] This successor to the NBISOM has an internal structure with lower area requirements combined with enhanced features. For instance, the best-match search is optimized. In spite of the integrated enhancements, the clock speed can be more than 40 MHz. The first NBX test chip integrates 16 processor elements, and each element holds 128 vector components with 8 bits per component in two static RAM banks with 128×64 bits. The chip is fabricated in a 0.8-micron CMOS process with a die size of 28.58 sq. mm.

Local cluster neural network

The LCNN is a two-layer feed-forward network developed by Sitte and Geva.[1] Similar to radial basis function networks, the LCNN uses local functions for approximating a given complex function. The approximation is done by placing several local functions in its input space. The basic idea is to superimpose sigmoid functions in such a way that they only respond to a finite region in input space. The network accomplishes this by first constructing a ridge function for every input signal x by comparing two sigmoids. The ridge function is: $l(w,r,k,x) = \sigma\,[k,w^T(x-r)+1] - \sigma\,[k,w^T(x-r)-1]$.

The orientation and width of the ridge is determined by the orientation of w and its length. The position of the ridge is given by the position of the vector r. The sigmoid is chosen to be the logistic function with steepness k: $\sigma(k,h) = 1/1 + e^{-kh}$. A local function is obtained by adding the ridge function of all the components in the input signal vector. All ridges have a different orientation but the same center:

$$f(w,r,k,x) = \sum_{i=1}^{n} l(w_i,r,k,x)$$

where w is now an $n \times n$ matrix made out of n ridge (column) vectors w_i. This function has a bump around the common center r and ridges emanating to infinity in as many directions as there are dimensions. These ridges have to be removed to make the function local. Application of a properly biased sigmoid σ_0 to the function $f(w,r,k,x)$ cuts out the ridges smoothly. The local function made out of a cluster of sigmoids is $L(w,r,k,x) = \sigma_0\,[f(w,r,k,x) - b]$.

A local cluster network consists of an array of local cluster functions all receiving the same inputs. The network output is a weighted sum of the local cluster outputs:

$$y(x) = \sum_{\mu=1}^{m} v_\mu L\left(W_\mu, r_\mu, x\right)$$

The LCNN makes use of the computational advantages of localized functions for training and application. It also has the advantage of relatively simple weighted sums and sigmoids for analog VLSI implementation, instead of calculating more complicated Euclidean distances.

References

1. S. Geva, K. Malmstrom, and J. Sitte, "Cluster Neural Net: Architecture, Training and Applications," *Neurocomputing 20*, 1998, pp. 35-56.

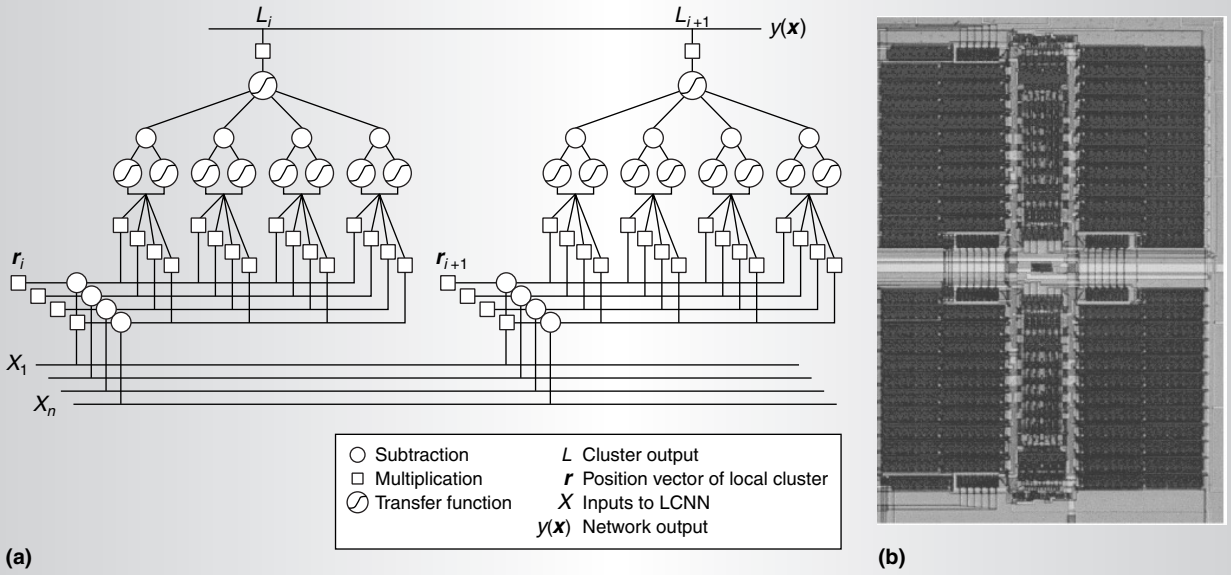


Figure 4. Architecture (a) and chip photograph (b) of a local cluster neural network.

Local cluster neural network

A third class of neural networks for hardware implementation is function approximators. This kind of network maps a given input vector from a continuous input space to a continuous output space. We examined both analog and digital hardware implementations of these network types, although in this article, we will examine only the analog approach. Other authors have discussed the digital implementation of a radial basis function network with an FPGA.[13]

The local cluster neural network (LCNN)[14] is a two-layer feed-forward network, which we implemented in analog hardware.[15] The network uses dot products and sigmoidal transfer functions, which are generated efficiently in analog hardware. See the "Local cluster neural network" sidebar for a detailed description of the operations of this network. Figure 4a shows the architecture of the analog LCNN. We implemented the LCNN using a 0.8-micron CMOS process from Austria Microsystems. The chip has a total area of 10 sq. mm, but the analog part only used 0.5 sq. mm. The rest of the area is used for the weights, which are stored in digital shift registers and converted with digital-to-analog converters. We placed two local clusters with six inputs and one output each on the chip, as shown in Figure 4b.

We chose current as the information carrier because most operations such as additions, subtractions, and multiplications can be implemented in current mode techniques. Furthermore, we can combine these circuits easily. The maximum output current is about 10 mA and the maximum input frequency of the chip is limited to 50 KHz for a supply voltage of 3 V. Control tasks, especially motor control in small autonomous systems, are the main application area of this LCNN chip. Therefore, we preferred low energy consumption instead of a high-frequency response.

Applications

Our proposed neural network hardware models offer a wide range of applications. The chips are suitable for use in information retrieval (associative memory), exploratory data analysis (self-organizing maps), automation, and autonomous systems.

Depending on the restrictions of a certain application, we can emphasize different aspects of the chip. First, we can operate in real time using neural hardware. Examples include automation with online identification and control using neural algorithms.[13] The NBISOM system offers advantages in the analysis of huge data sets. This system offers a significant computation time speed-up compared with a software implementation; not only in self-organizing networks, but also compared with conventional methods (cluster analysis for example).

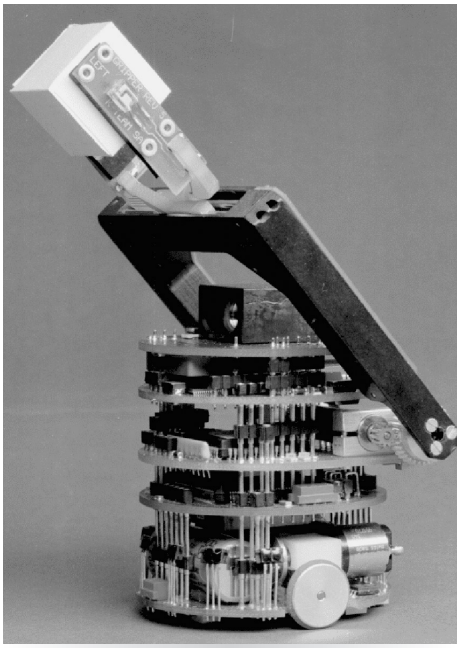


Figure 5. The minirobot Khepera.

Second, our hardware allows the development of resource-efficient systems with minimal total energy consumption combined with a small size and fault-tolerant behavior. To examine the use of neural concepts in autonomous systems, we embedded the developed neural hardware in the minirobot Khepera (shown in Figure 5) from K-Team, Switzerland.[16]

The design goal for autonomous systems is the generation of reasonable reflex-like behavior, based on the available sensor data. In particular, for sensor calibration we embedded concepts and methods of self-organizing feature maps into the robot's system control structure.[16] The robot can adjust sensor deviations (inevitably caused during manufacture) in a self-optimizing way. We used associative memory for user-adaptive and fault-tolerant behavior mapping. In this application, a sensor input acts as the input vector for the memory matrix. The associated output is interpreted as motor stimuli to achieve an adequate motion.

Benefits

There are three main reasons for the concentration on model-specific implementa-

tions such as those we have examined. First, because of the modular and regular architecture of the selected ANN models, the microelectronic implementation can be performed efficiently. Second, only a small amount of additional software is necessary to embed these chips in a given system environment. This is crucial for both hardware platforms used in our laboratory, that is, the stationary VMEbus/PCIbus system for simulating large ANNs in real time and the mobile autonomous robot Khepera. Third, special-purpose hardware ANN implementations are efficient in speed, area, and power consumption.

We made the ANN solutions resource efficient for suitability in real-time applications with extremely low energy requirements, and even for the smallest device sizes. Furthermore, due to the modular architecture, our chip implementations are easily adapted to new technologies without enormous design and test complexity. Thus, these architectures will benefit more from future developments in microelectronics than would software microprocessor solutions. Additionally, the system features (storage capacity, classification quality, approximation accuracy) improve noticeably as the size of the network increases.

Progress in microelectronics provides a powerful framework for implementing large ANNs in hardware. Microelectronics will still dominate the field of ANN implementation at least for the next decade. However, the discussion is open about the best ways to achieve very large neural systems and, in the long term, how to produce so-called artificial brains. We are still a long way from fully comprehending the functional mechanisms of the brain; and the construction of an artificial brain will remain for a very long time, if not forever, a fantasy. Nevertheless, we do have something to learn from nature about resource-efficient technical systems. This is why a hardware realization of neural networks does not aim for an exact reproduction of nervous systems, but simply for the efficient use of technologies for solving technical problems. Today we use microelectronics; we are keenly awaiting the technology we can use tomorrow. MICRO

Acknowledgment

This work was partly supported by the Deutsche Forschungsgemeinschaft (German Research Council) DFG GR 948/14-3, DFG RU 477/2-3 and the Graduate Centre Parallele Rechnernetzwerke in der Produktinstechnik.

References

1. U. Ramacher and U. Rückert, eds., *VLSI Design of Neural Networks*, Kluwer Academic, Boston, 1991.
2. *IEEE Micro* (Special Issue on Approximating Solutions: Fuzzy Systems and Neural Networks), vol. 15, no. 3, June 1995.
3. P. Inne, "Digital Connectionist Hardware: Current Problems and Future Challenges," *Biological and Artificial Computation: From Neuroscience to Technology*, *Lecture Notes in Computer Science*, vol. 1240, Springer, Berlin, 1997, pp. 688-713.
4. G. Palm et al., "Neural Associative Memories," *Associative Processing and Processors*, A. Krikelis and C.C. Weems, eds., IEEE CS Press, Los Alamitos, Calif., 1997, pp. 307-326.
5. U. Rückert, A. Funke, and C. Pintaske, "Acceleratorboard for Neural Associative Memories," *Neurocomputing 5*, 1993, pp. 39-49.
6. U. Rückert, "An Associative Memory with Neural Architecture and its VLSI Implementation," *Proc. Hawaii Int'l Conf. System Sciences* (HICSS-24), IEEE CS Press, Los Alamitos, Calif., 1991, pp. 212-218.
7. A. Heittmann et al., "Digital VLSI Implementation of a Neural Associative Memory," *Proc. 6th Int'l Conf. Microelectronics for Neural Networks, Evolutionary and Fuzzy Systems*, Univ. of Technology, Dresden, Germany, 1997, pp. 280-285.
8. A. Heittmann and U. Rückert, "Mixed Mode VLSI Implementation of a Neural Associative Memory," *Proc. 7th Int'l Conf. Microelectronics for Neural Networks, Evolutionary and Fuzzy Systems*, Univ. of Technology, Dresden, Germany, 1999, pp. 205-211.
9. T. Kohonen, *Self-Organizing Maps,* Springer, Berlin, 1995.
10. S. Rüping, K. Goser, and U. Rückert, "A Chip for Self-Organizing Feature Maps," *IEEE Micro*, vol. 15, no. 3, 1995, pp. 57-59.
11. S. Rüping, M. Porrmann, and U Rückert,

"SOM Hardware-Accelerator," *Proc. Workshop Self-Organizing Maps* (WSOM 97), Helsinki Univ. of Technology, Helsinki, Finland, 1997, pp. 136-141.
12. S. Rüping, M. Porrmann, and U Rückert, "SOM Hardware with Acceleration Module for Graphical Representation of the Learning Process," *Proc. 7th Int'l Conf. Microelectronics for Neural Networks, Evolutionary and Fuzzy Systems*, 1999, pp. 380-386.
13. U. Witkowski et al., "System Identification Using Self-Organizing Feature Maps," *Proc. 5th Int'l Conf. Artificial Neural Networks* (ANN 97), IEEE CS Press, Los Alamitos, Calif., 1997, pp. 100-105.
14. S. Geva, K. Malmstrom, and J. Sitte, "Cluster Neural Net: Architecture, Training and Applications," *Neurocomputing 20*, 1998, pp. 35-56.
15. T. Körner, *Analog VLSI Implementation of a Local Cluster Neural Net*, doctoral dissertation, no. 77, Heinz Nixdorf Inst., Univ. of Paderborn, Germany, 2000.
16. U. Rückert, J. Sitte, and U. Witkowski, "Autonomous Minirobots for Research and Edutainment," *Proc. 5th Int'l Heinz Nixdorf Symp.*, no. 97, Heinz Nixdorf Inst., Paderborn, Germany, 2001.

Ulrich Rückert is a professor of electrical engineering and holds the System and Circuit Technology chair at the Heinz Nixdorf Institute at the University of Paderborn, Germany. His research interests include the design of microelectronic systems for massively parallel and resource-efficient information processing. Rückert received the diploma degree (M.Sc.) in computer science and the Dr. -Ing. degree in electrical engineering from the University of Dortmund, Germany. He is a member of the IEEE and the International Neural Network Society.

Direct questions and comments about this article to Ulrich Rückert, Heinz Nixdorf Institute, University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany; rueckert@hni.upb.de.