TECHNICAL REPORT NO. 313

An Analog VLSI Array Processor For Classical and Connectionist AI

by

Jonathan W. Mills and Charles A. Daffinger

July 1990

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY

Bloomington, Indiana 47405-4101

AN ANALOG VLSI ARRAY PROCESSOR FOR CLASSICAL AND CONNECTIONIST AI

JONATHAN WAYNE MILLS*
CHARLES A. DAFFINGER†

Indiana University
Bloomington, Indiana 47405

Abstract

Łukasiewicz logic arrays (ŁLAs) are analog VLSI array processors based on an 11-transistor cell that possesses a dual algebraic and logical semantics. By switching between semantics ŁLAs support both classical and connectionist artificial intelligence applications. In this paper we briefly describe ŁLA architectures, CMOS VLSI implementation, and programming. Data resulting from the use of the prototype ŁL9 as a fuzzy function recognizer is reported. The application of ŁLAs to expert systems, analog models of single neurons, and simple recurrent neural networks is also described.

1. INTRODUCTION

This paper describes the implementation and applications of Łukasiewicz logic arrays (ŁLAs). ŁLAs are analog VLSI array processors based on a processing element that possesses a dual algebraic and logical semantics. By switching between semantics this architecture supports both classical and connectionist artificial intelligence applications.

1.1 Classical and Connectionist AI

Artificial intelligence (AI) applications are broadly categorized as either:

- classical (symbolic), typified by theorem proving, logic programming and expert systems, or
- connectionist (sub-symbolic), typified by pattern classification and learning algorithms, with little or no requirement for logical inference or explicit representation of a problem domain.

---

* 101 Lindley Hall, Department of Computer Science, (812) 855-6486, jwmills@iuvax.cs.indiana.edu

† 101 Lindley Hall, Department of Computer Science, (812) 855-6486, cdaf@iuvax.cs.indiana.edu

Application-specific architectures for AI are similarly categorized: architectures for classical applications provide hardware data types, unification support and control for imperative search strategies, but have few or no features for massively parallel pattern classification; architectures for connectionist applications provide massive parallelism and local communication between processors, and favor rapidly reconfigurable weights or threshold values whose precision is often less than six bits.

The need for parallelism in classical AI applications, and symbolic processing in connectionist AI applications has led to research into hybrid symbolic and sub-symbolic architectures [1, 2, 3].

1.2 Digital, Analog and Hybrid Computers

The differences between analog and digital architectures parallel the differences between AI architectures. Digital computer architectures may be classified as either general purpose, such as RISC and CISC architectures, or application specific, such as systolic arrays or language processors. Properties that have made digital computers the dominant form of computing device include the ability to handle discrete data, arbitrary precision, programmed control, and the availability of dense program and data stores.

Analog computers provide an analogy to a physical system. An analog computer need not be electronic, but could be composed of soap bubbles, conductive paper, strings and weights, pins and wires, or scale models of aircraft. In fact, all of these have been used with varying success as analog computers [4]. The advantages of analog computers include their speed, inherent parallelism, and small size. For example, to study melting Murray [5] has implemented an analog computer with colloidal crystals that uses 10^9 "processors" in an area of approximately 3 cm^2. To study this same problem with a general purpose digital computer would require 15,000 Connection Machines, each with 65K processing elements. As well-suited as Murray's computer is to its task, it also exhibits the disadvantages of analog computers. These include limited accuracy, inability to process discrete data, and applicability to a small class of problems.

In a similar fashion ŁLAs have limitations that can be avoided by including ŁLAs in hybrid digital-analog computer architectures: for example, although capacitance can be designed into an ŁLA to store analog values for a network's weights in the short term, digital storage is better-suited to store weights for repeated long-term use.

Another type of hybrid architecture exists when computers possess different operational semantics at a fundamental level, allowing the operation of the processor to change without modifying the hardware or programming an emulator or interpreter are also "hybrid" architectures. Architectures that have this property are not paradigm-specific, and can support fuzzy logic [6] as well as neural networks [7, 8, 9].

ŁLAs both possess the second type of hybridism, and can be used as processors in hybrid digital-analog computers; thus ŁLAs are well-suited as "bridge" architectures that span the gap between digital and analog processors, and classical and connectionist AI applications.

2. ŁUKASIEWICZ LOGIC ARRAYS

2.1 Design

Systolic logic arrays (SLAs) implement sentence schema of multiple-valued logics with small, fast, analog processing elements that have limited precision. *Łukasiewicz logic arrays*, or *ŁLAs* are systolic logic arrays restricted to sentence schema of the multiple-valued Łukasiewicz logic [10, 11]. ŁLAs are organized as H-trees of identical processing elements. A processing element may perform Łukasiewicz implication ($\rightarrow$), negated implication ($\not\rightarrow$), or both, with the connectives defined as follows:

$$\alpha \rightarrow \beta \quad \equiv \quad min(\, 1,\, 1 - \alpha + \beta \,)$$

$$\alpha \not\rightarrow \beta \quad \equiv \quad \neg\, (\alpha \rightarrow \beta) \quad \equiv \quad max(0,\, \alpha - \beta)$$

The architecture is implemented as an analog VLSI array of processing elements (PEs), with each PE composed of only 11 MOSFET transistors. The first 32-PE array prototype ŁL9 was fabricated on a MOSIS "tiny" chip in January, 1990; the second prototype, ŁL10, in June, 1990.

The design of ŁL9 and ŁL10 followed from the relation of Łukasiewicz logic to the theory of cellular automata, and inference cellular automata in particular [12]. An ŁLA processor is an H-tree interconnection network whose nodes are processing elements that perform simple — even single — functions: Łukasiewicz implication ($\rightarrow$), negated Łukasiewicz implication ($\not\rightarrow$), or both. The processing elements are derived from the bounded difference circuit of Takeshi Yamakawa [13, 14], leading to an architecture that is simple, regular and pipelined: a systolic array for Łukasiewicz logic.

The operation of the prototypes ŁL9 and ŁL10 is described elsewhere [33].

2.2 Dual Operational Semantics of ŁLAs

Łukasiewicz' logic is sufficiently general to deal with a broad class of approximate reasoning paradigms because it is representationally complete relative to the class of multiple-valued logics whose valuation functions can be defined in terms of +, −, *min* and *max*, including fuzzy logic [15, 16, 17, 18, 19].

McNaughton's theorem [20] allows us to use Ł at different levels of abstraction, in particular as a classifier for elements of fuzzy sets. By showing that valuation functions for connectives in sentences in Ł are equivalent to piecewise-constructable first-degree polynomials that map the hyperspace $[0,1]^n$ into the interval [0,1], the capability of building fuzzy pattern recognizers is provided. Thus, a series of sentences in Ł defines the polytope of some convex and simply connected solid in a hyperspace of degree n. This allows us to express arbitrarily complex membership relations in logical form; in a VLSI circuit we define the polytope with a sentence from Ł, which is converted to the normal form (described in section 3) of the sentence representable by one or more ŁLA circuits. The ŁLA circuit may be "programmed" to deal with variants of the original sentence by assigning incoming data to specific circuit inputs.

The dual semantics gives Łukasiewicz logic arrays surprising utility. Within the array some processing elements may operate at a low level of abstraction, evaluating membership

in a fuzzy set, while other processing elements may operate at a higher level of abstraction, implementing connectives of fuzzy logic. A single Łukasiewicz logic array can implement an expert system by processing input from analog sensors (fuzzy membership), accepting judgmental input in the form of confidence factors, then identifying a likely problem and its solution by a fuzzy logical deduction. It is our hypothesis that the Łukasiewicz logic array is a fundamental processor for approximate reasoning systems for two reasons: first, because the Łukasiewicz logic is complete with respect to the class of multiple-valued logics $L_{\{+,-,\wedge,\vee\}}$ as shown in this paper; and second because the dual semantics of Ł allow it to describe expert systems, Boltzmann machines [7], neural networks [7, 21], fuzzy computers [6, 22], and sparse distributed memories [23].

2.3 Advantages and Disadvantages of ŁLAs as AI Processors

ŁLAs have several advantages as AI processors: they can implement both classical and connectionist AI systems by switching between the dual logical and algebraic semantics of Łukasiewicz logic. Łukasiewicz logic can be used in a logical sense to model propositions whose truth values are distributed over the interval [0,1], thus Łukasiewicz logic is closely related to fuzzy logic [16, 19, 22, 24, 25]. Yamakawa's designs for fuzzy inference engines and expert systems use primitive functions equivalent to the connectives of Łukasiewicz logic [6, 13, 26].

The homogeneous arrays used in the prototype ŁLAs require $O(2^n)$ nodes to implement a sentence containing n logical variables. One approach to this problem is the development of heterogeneous arrays that do not use binary tree form for sentence schemata. A second approach is to time-multiplex the inputs, and recursively evaluate each sentence. This reduces the number of nodes to O(n).

The precision of analog ŁLAs limits their accuracy, and factors such as stabilization time and environmental factors are difficult to control. However, the fuzzy nature of the computations reduces the seriousness of this problem, and first test results show accuracy that ranges from 0.5% to 2% after trimming.

Finally, ŁLA programming is an instance of the more general problem of programming analog and hybrid computer architectures. We describe the low-level ŁLA programming methodology in the following section.

3. PROGRAMMING ŁLAS

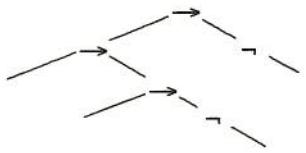
ŁLAs are programmed at the lowest level by fixing an interconnection network for the inputs, and presenting inputs that are either *true*, *false*, or variable. Because it is not practical to build an ŁLA for each sentence in Ł, it is necessary to develop a normal form that maps arbitrary sentences onto some general ŁLA.

The prototype ŁLA is structured as a binary tree whose nodes are connectives and whose leaves are logical variables. Most sentences in Ł do not map directly to this schema, but must be transformed to equivalent sentences which do. This general form of a sentence in Ł is the balanced normal form in implication, with explicit negation possible anywhere in the sentence.

Definition 1. *A sentence in Ł is in balanced normal form in implication if there exists some designated implication in the sentence, starting at which a binary tree of implications can be extracted, and for which at each non-leaf node in the tree the number of implications and logical variables in each subtree rooted at that node is equal.*

The circuit implements balanced normal form sentences in Ł because it is structured as an H-tree. The use of a binary tree to realize n-input R-valued functions for multiple-valued logic circuits was described by [27].

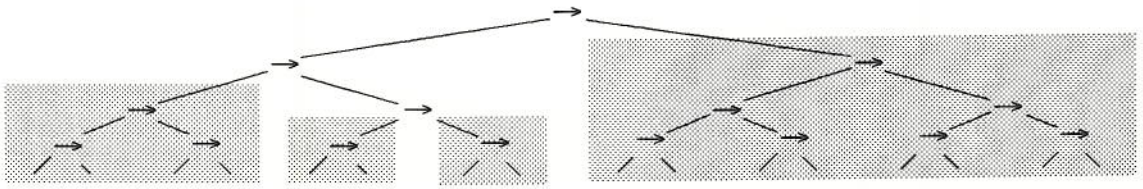
The next step toward developing a useful normal form is to remove explicit negation. Simplification will suffice to remove an occasional pair of negations, but single negated expressions of the form $\neg\alpha$ must be re-written to an equivalent form that does not use negation explicitly, namely $\alpha \rightarrow false$. The advantage is that clauses expressed in only one connective, while textually more complex, may be mapped to smaller and simpler physical devices that perform negation using data inputs alone. Consider the transformation of an arbitrary sentence in Ł to BNF normal form. The sentence is unbalanced initially, and contains negation (Figure 1).



$$(\alpha \rightarrow (\beta \rightarrow \neg\gamma)) \rightarrow \neg\delta$$

Figure 1. Unbalanced sentence in Ł

The resultant BNF normal form to which it is transformed is shown next (Figure 2). Although the textual form of the sentence is more complex, the BNF normal form uses cells of the ŁLA that the first form would have left unused. These "extra" inputs and implications can be used to adjust the constraints under which the sentence is true.



$$(((T \rightarrow T) \rightarrow (T \rightarrow \alpha)) \rightarrow ((T \rightarrow \beta) \rightarrow (\gamma \rightarrow F))) \rightarrow (((T \rightarrow T) \rightarrow (T \rightarrow \delta)) \rightarrow ((T \rightarrow T) \rightarrow (T \rightarrow F)))$$

Figure 2. BNF normal form of sentence in Figure 1

This programming methodology is useful to define membership functions for fuzzy logic, but much less so for designing neural networks. However, it appears that the application of ŁLAs to neural networks may not require this approach: a network built based on ŁLAs using randomly selected inputs for pattern data, others for training, and the remainder for feedback strongly resembles a simple recurrent network.

4. CLASSICAL AI APPLICATIONS

Łukasiewicz logic arrays were first proposed to evaluate sentences in Ł, but because Łukasiewicz logic describes other forms of approximate reasoning, ŁLAs are useful for a variety of applications. The dual logical and algebraic semantics of Ł allow ŁLAs to implement expert systems, neural networks [7, 21], and fuzzy computers [6, 22]. We present schematic examples for each application, and report the results obtained by programming the prototype ŁLA as a fuzzy function generator.

4.1 Fuzzy Logic

Łukasiewicz logic is closely related to fuzzy logic [16, 19, 22, 24, 25]. Yamakawa shows designs for fuzzy inference engines and expert systems which may be embedded in Łukasiewicz logic arrays [6, 13, 26]. We have used the prototype ŁLA to compute fuzzy membership functions, and present the results obtained along with observations on the error measured after trimming the output of the prototype ŁLA.

The first function we evaluated was the "half notch" (Figure 3). The untrimmed ŁLA has a stable 10% output error when implementing this function, which can be removed easily.

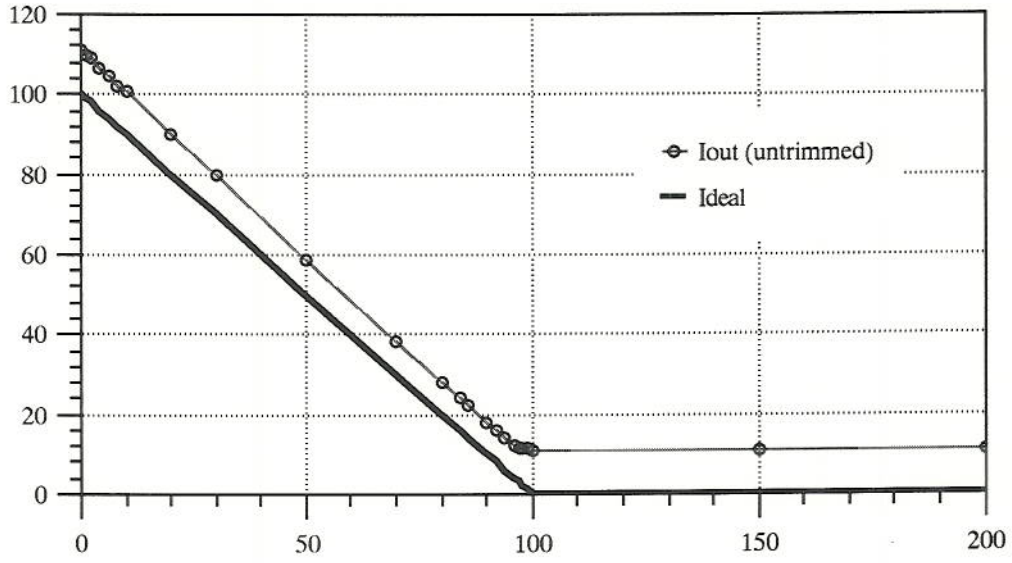


Figure 3. ŁLA implementation of ((¬ α → α) → F

The next function (Figure 4), an inverted "notch," is defined by the expression (((¬ α → α) → ¬ (α → ¬ α)) → β). This membership function was programmed into the 31-cell ŁLA as the following 32-element vector:

$$(F,F,F,F,\alpha,F,T,\alpha,T,\alpha,\alpha,F,F,F,T,F,F,F,F,F,F,F,F,F,F,F,F,F,F,F,T,\beta).$$

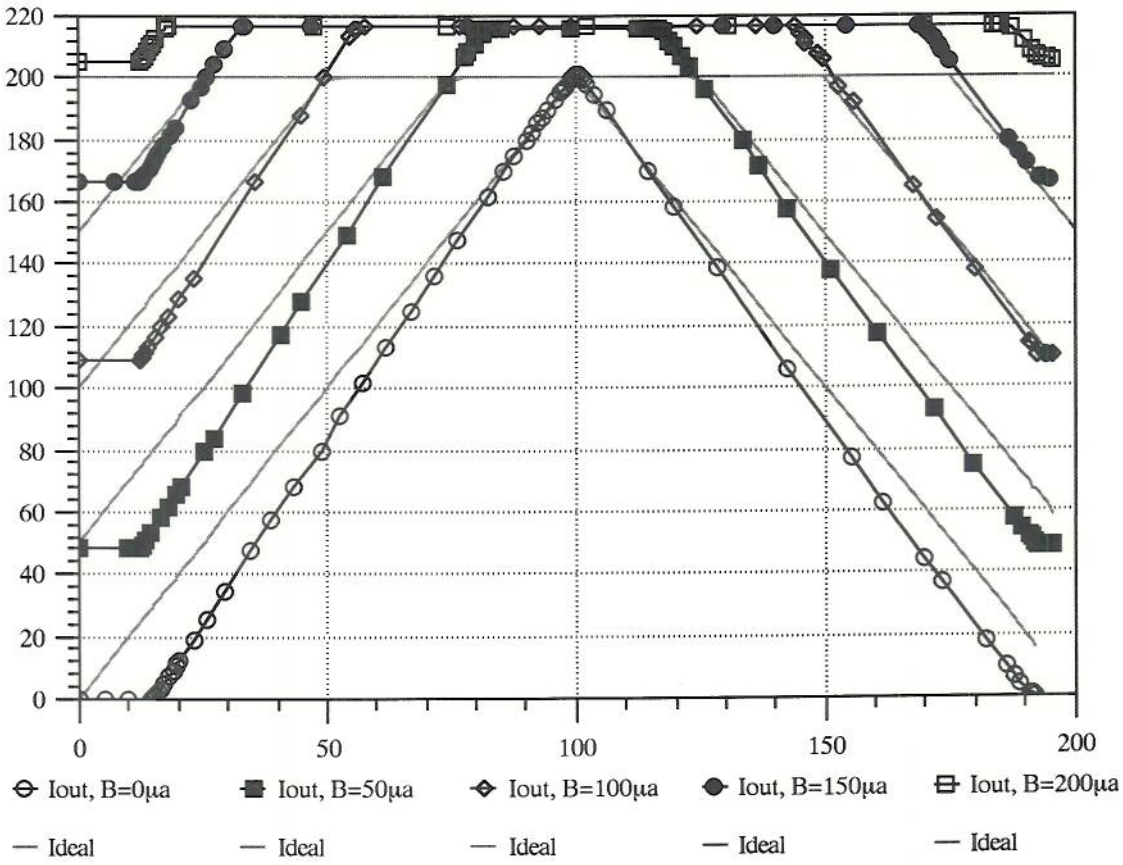


Figure 4. ŁLA implementation of ((¬ α → α) → ¬ (α → ¬ α)) → β

This function can be turned into a series of "clipped notches" by instantiating β to different truth values. The worst-case error in this function is approximately 14%, but can be reduced by trimming [28, 29].

The final fuzzy membership function is the uninverted "notch", defined by the expression (¬ α → α) → ¬ (α → ¬ α). This membership function was programmed into the 31-cell ŁLA as the following 32-element vector:

$$(T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,T,\alpha,F,T,\alpha,T,\alpha,\alpha,F,T,T,T,F).$$

The result of this experiment showed that the ŁLA implemented the notch function linearly, but with a slope that varied from that of the calculated function. This scaling is due to an arbitrary choice of a resistor in the measuring circuit. By changing this resistance, or "trimming", the output can be adjusted to produce a much closer fit, with the center of the notch near 100μA (Figure 5). We have calculated that an optimally trimmed ŁLA would have a typical error of less than 2%, and a mean error less than 0.5%. The ŁLA computes fuzzy functions accurately. Error is within 1.5% of ideal, with error near 4% where the current changes value rapidly.

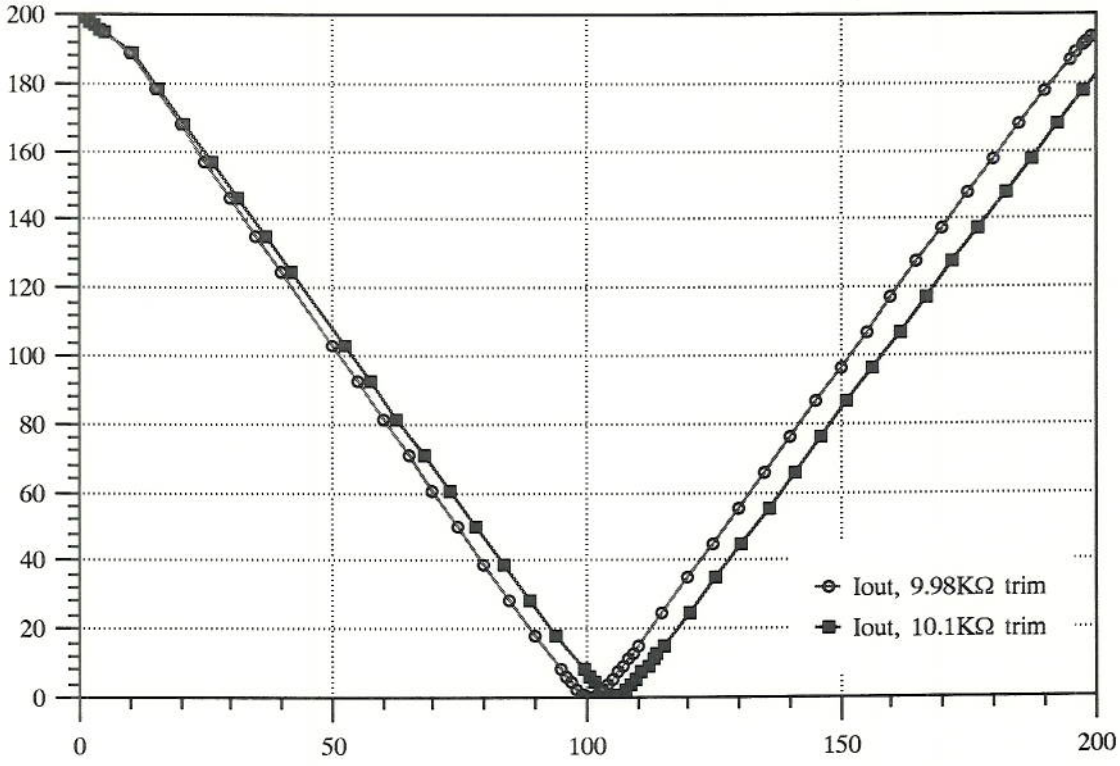


Figure 5. ŁLA implementation of (¬ α → α) → ¬ (α → ¬ α)

4.2 Expert Systems

ŁLAs implement expert systems by mapping membership functions to processing elements at lower levels in the array, and rules to processing elements higher in the array. A rule is a single tree that is true or false to a degree that depends on its inputs. Rules can be designed that do not fire unless their inputs reach a desired confidence level (Figure 6).

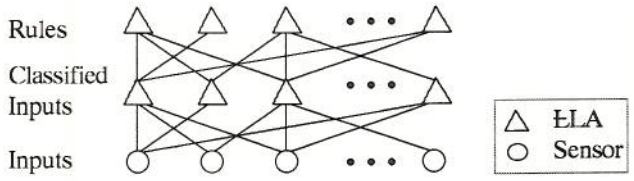


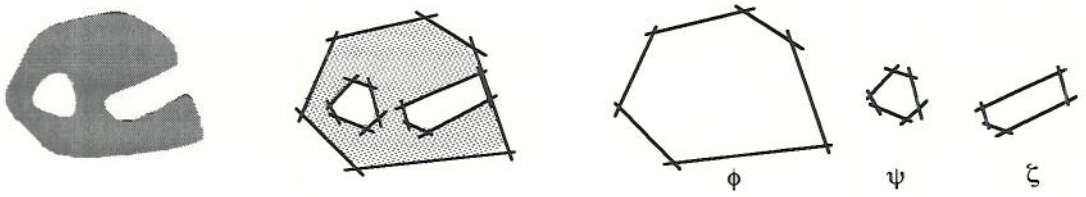
Figure 6. ŁLA implementation of expert system

A single Łukasiewicz logic array can implement a simple expert system, which may be used to embed limited "intelligence" in individual sensors. Within the array some processing elements operate at a low level of abstraction, evaluating membership in a fuzzy set, while other processing elements operate at a higher level of abstraction, implementing a rule for that sensor. The rule's operation may vary based on control inputs to the ŁLA. The expert system evaluates its sensor's input, firing the rule if the confidence factor is exceeded.

5. CONNECTIONIST AI APPLICATIONS

McNaughton's theorem (see Section 2) and Giles' Logic of Assertions [25] relate sentences in Łukasiewicz logic to piecewise-linear functions and the theory of convex analysis. This is the functional domain of pattern recognizers and classifiers (Figure 7), which has encouraged us to investigate neural networks implemented with ŁLAs.

Łukasiewicz logic also has an algebraic interpretation. The claim that Łukasiewicz logic arrays are useful as pattern recognizers and classifiers is due to a theorem of McNaughton. This theorem relates sentences in Łukasiewicz logic to piecewise-linear functions [20]. A more recent observation by Giles points out the similarity between resolution using his Logic of Assertions (LA) and the theory of convex analysis — linear spaces with convex sets and convex functions [25]. This is exactly the functional domain of perceptrons and neural networks, strongly suggesting that ŁLAs can be used to emulate the pattern classification functions of neural networks (Figure 7).



Non-convex space | Decomposed into convex hulls | Expressed as a sentence in Lukasiewicz logic: $\phi \wedge \neg \psi \wedge \neg \zeta$

Figure 7. Relationship of non-convex space to a sentence in Łukasiewicz logic

When symbolic and sub-symbolic computation are merged in an ŁLA, some processing elements operate at a low level of abstraction evaluating membership in a category, while other processing elements operate at a higher level of abstraction evaluating the truth of a logical sentence.

Early models of nerve nets were described by McCulloch and Pitts [30]. Kleene and von Neumann anticipated much of the present-day work in neural networks, offering theoretical descriptions of the events representable in neural networks [31], and the creation of reliable computing systems from unreliable components [32].

5.1 Model of a Neuron

The evaluation formula for Łukasiewicz implication shows how it may be used to construct a very simple "neuron." In the expression $min(1, 1 - \alpha + \beta)$, α is an inhibitory input that lowers the "firing rate", or truth value. β is an excitatory input that increases the "firing rate." Recursively connecting several implication cells produces a "neuron" with a variable threshold (Figure 8a). Summation units can also be devised (Figure 8b).

Simulations of interconnected ŁLA "neurons" and summing elements show that they have the basic properties needed to construct a neural network. The behavior of a "neuron" can be changed by modifying its threshold. For example, the slight delay before the second output pulse in the simulation is due to an intermediate "neuron" (Figure 9).

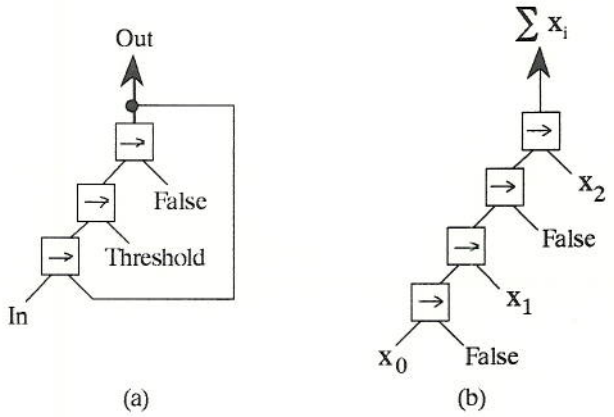


Figure 8. Implementing a "neuron" and summing element with an LLA

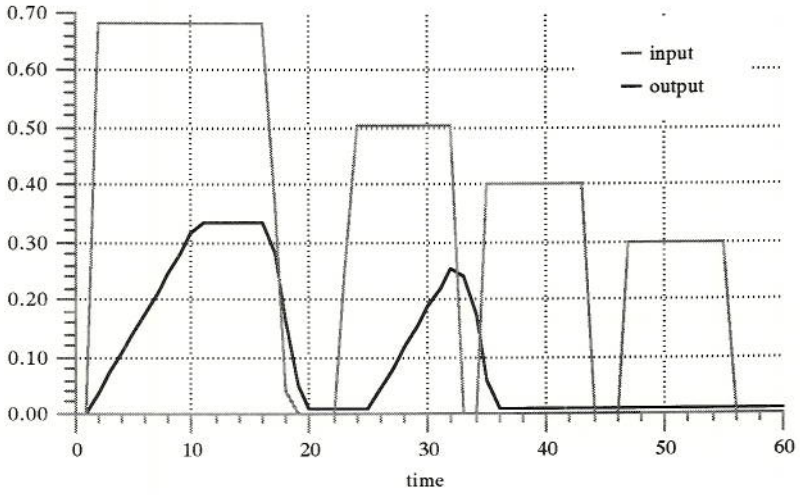


Figure 9. Simulation of interconnected LLA "neurons"

We are now working to construct a trainable neural network from these basic components. The initial version will be a hybrid digital-analog system, similar to those of Alspector and Graf [7, 8, 9]. However, we hope to devise analog-only systems using double-poly capacitors as storage elements for weights.

5.2 Simple Recurrent Networks

LLAs provide a forest of sentences all of which may be true to various degrees depending on order of inputs. A programmed LLA defines a metric on a set of inputs, expressed as the truth of a logical sentence. Thus a neural net can be built with LLAs by using a large sentence, some of whose inputs are mapped to statistically separable patterns, and the rest mapped to weights adjusted during the learning process. Learning in an LLA is a process of generating a sentence that is maximally true only for elements of the training set, and false for others. Due to the analog nature of LLAs, the less similar a new element is to those in the training set, the less likely it is to be recognized.

We close with a brief summary of our results and directions for future research.

6. CONCLUSIONS

We described the architecture of an operational 31-cell CMOS VLSI ŁLA, which is regular, simple, area-efficient and implemented with analog rather than digital processing elements. The prototype ŁLAs are programmed with input vectors derived from normal forms of sentences in the Łukasiewicz logic. This requires data inputs on the order of $O(2^n)$ for sentences in n implications, limits the size of the sentences that can be evaluated by a given ŁLA and increases the number of pins needed on the VLSI package. However, many data inputs are *true* or *false*, or are composed of a repeated number of variable inputs. Based on this observation we are designing ŁLAs that have external control inputs, and a restricted number of external data inputs. ŁLA programming is an instance of the more general problem of programming analog and hybrid digital-analog computer architectures. Because ŁLA-based systems will be either analog or hybrid digital-analog computers, future research includes developing programming languages for them.

The dual logical and algebraic semantics of Łukasiewicz logic allow ŁLAs to implement expert systems, neural networks, and fuzzy logic functions. We presented schematic examples for each application, and reported the results obtained by programming the prototype ŁLA as a fuzzy function generator. The results showed that the ŁLA implemented the notch function linearly, but with a slope that varied from that of the calculated function. Trimming the ŁLA inputs and outputs is expected to result in a typical error of less than 2%, and a mean error less than 0.5%. With further modifications to the circuit, and trimmed outputs, the error should drop to less than 1% for each cell as reported by Yamakawa [13].

Applications of ŁLAs to expert systems, a model neuron, and simple recurrent networks were proposed. Current work to complete a hybrid digital-analog testing environment for ŁLAs with up to 128 inputs will allow us to implement the proposed applications using the prototype ŁLAs. ŁLAs present a new challenge in the design of massively parallel processors, as well as the design and programming of analog and hybrid computers. For those problems where precision may be traded for speed, ŁLAs provide an excellent solution.

[1] Hendler, J. A. 1989. *A Hybrid Connectionist/Symbolic Model*. ONR 1988 Program Summary.

[2] Fagin, B. 1989. *"What can be done with 64,000 processors?"*. Talk delivered at 1989 North American Conference on Logic Programming, Parallel Logic Programming Workshop.

[3] Touretzky, D. November 9, 1989. *Colloquium titled: "Getting Beyond Back-Prop: What Connectionism Really Offers AI"*.

[4] Karplus, W. 1956. *Analog Simulation*.

[5] Murray, C. 1990. *Colloidal crystals as analog computers*.

[6] Yamakawa, T. 1988. High-speed fuzzy controller hardware system: The Mega-FIPS machine. *Information Sciences* **45** (2): pp. 113-128.

[7] Alspector, J., and R. B. Allen. 1987. *A neuromorphic VLSI learning system*. In *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*. Edited by P. Losleben. pp. 313-349. Cambridge, Massachusetts: MIT Press.

[8] Alspector, J., R. B. Allen, V. Hu, and S. Satyanarayana. 1987. Stochastic learning networks and their electronic implementation. *Proceedings of Neural Information Processing Systems—Natural and Synthetic*. Denver, Colorado, November 8-12:

[9] Graf, H. P., L. D. Jackel, and W. E. Hubbard. 1988. VLSI implementation of a neural network model. *IEEE Computer* **21** (3): pp. 41-49.

[10] Kung, H. T., and C. E. Leiserson. 1978. Systolic arrays (for VLSI). *Proceedings of Symposium on Sparse Matrices Computations*. Knoxville, Tennessee: SIAM. pp. 245-282.

[11] Lukasiewicz, J., and A. Tarski. 1930. Untersuchungen über den Aussagenkalkül. *Comptes rendus des séances de la Société des sciences et des lettres des Varsovie Classe III* (23): pp. 30-50.

[12] Mills, J. W. 1989. *Inference Cellular Automata*. (In preparation).

[13] Yamakawa, T., and T. Miki. 1986. The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process. *IEEE Transactions on on Computers* **C-35** (2): pp. 161-167.

[14] Yamakawa, T., T. Miki, and F. Ueno. 1985. The design and fabrication of the current mode fuzzy semi-custom IC in the standard CMOS IC technology. *Proceedings of IEEE 17th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 76-82.

[15] Gaines, B. R. 1976. Fuzzy reasoning and the logics of uncertainty. *Proceedings of IEEE 6th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 179-188.

[16] Giles, R. 1979. *A formal system for fuzzy reasoning*. In *Fuzzy Sets and Systems 2*. pp. 233-257. North-Holland.

[17] Katz, M. 1981. Two systems of multi-valued logic for science. *Proceedings of IEEE 11th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 175-182.

[18] Katz, M. 1982. Real-valued models with metric equality and uniformly continuous predicates. *Journal of Symbolic Logic* **47** (4): pp. 772-792.

[19] Zadeh, L. A. 1975. Fuzzy logic and approximate reasoning. *Synthese* **30** pp. 407-428.

[20] McNaughton, R. 1951. A theorem about infinite-valued sentential logic. *Journal of Symbolic Logic* **16** pp. 1-13.

[21] Mattrey, R. F., D. D. Givone, and C. M. Allen. 1973. Applying multiple-valued algebra concepts to neural modeling. *Proceedings of IEEE International Symposium on Multiple-Valued Logic*. Toronto, Canada:

[22] Giles, R. 1976. Lukasiewicz logic and fuzzy set theory. *Int. J. Man-Machine Studies* **8** pp. 313-327.

[23] Kanerva, P. 1988. *Sparse Distributed Memory*. Cambridge, Massachusetts: MIT Press.

[24] Giles, R. 1982. Semantics for fuzzy reasoning. *Int. J. Man-Machine Studies* **17** pp. 401-415.

[25] Giles, R. 1985. A resolution logic for fuzzy reasoning. *Proceedings of IEEE 17th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press. pp. 60-67.

[26] Yamakawa, T., and H. Kabuo. 1988. A programmable fuzzifier integrated circuit—synthesis, design and fabrication. *Information Sciences* **45** (2): pp. 75-112.

[27] Hurst, S., L. 1986. A survey: developments in optoelectronics and its applicability to multiple-valued logic. *Proceedings of IEEE 16th Symposium on Multiple-Valued Logic*. Blacksburg, Virginia: IEEE Computer Society Press. pp. 179-188.

[28] Mills, J. W., M. G. Beavers, and C. A. Daffinger. 1989. Lukasiewicz Logic Arrays. *Technical Report (in preparation)*

[29] Mills, J., and C. Daffinger. 1990. CMOS VLSI Lukasiewicz Logic Arrays. *Proceedings of Application Specific Array Processors*. (accepted). Princeton, New Jersey:

[30] McCulloch, W. S., and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5** pp. 115-133.

[31] Kleene, S. C. 1956. *Representation of events in nerve nets and finite automata*. In *Automata Studies*. Edited by C. E. Shannon and J. McCarthy. pp. 3-41. Princeton: Princeton University Press.

[32] von Neumann, J. 1956. *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*. In *Automata Studies*. Edited by C. E. Shannon and J. McCarthy. pp. 43-99. Princeton, New Jersey: Princeton University Press.

[33] Mills, J., and C. Daffinger. 1990. CMOS VLSI Lukasiewicz Logic Arrays. *Proceedings of Application Specific Array Processors*. (accepted). Princeton, New Jersey: