Reviews of Books and Papers in the Computer Field

DONALD L. EPLEY, REVIEWS EDITOR

D. W. FIFE, A. J. NICHOLS, A. I. RUBIN, H. S. STONE
ASSISTANT REVIEWS EDITORS

Please address your comments and suggestions to the Reviews Editor: Donald L. Epley, Department of Electrical Engineering, University of Iowa, Iowa City, Iowa 52240

A. SYSTEMS

R67-51 Electronic Digital Systems—R. K. Richards (New York: Wiley, 1966).

The heart of this book is the extensive chapter on stored program machines which appears to have been a reasonably successful attempt to speak meaningfully to every significant nonarithmetic, nonphysical design concept in that area. Other major chapters include: an excellent history of electronic digital computers (which the author terms conventional except for its references to the work of Atanasoff and Berry at Iowa State University in the early 1940's); a straightforward presentation of Mealy-Moore sequential circuits; a brief introduction to automatic programming; and a generally good discussion of information theoretic, modulation, and error control aspects of digital data transmission. The remainder of the book is devoted to hybrid systems, telephone and message switching, reliability analysis, automatic digital system design, analogies to human thought processes, and miscellaneous digital systems.

Dr. Richards displays wide knowledge in the areas covered and, for the most part, he has succeeded in his avowed purpose of presenting extracted and organized concepts rather than a myriad of details. Practical aspects are stressed strongly and sound engineering appraisals abound—refreshingly so. The book also contains an excellent set of well categorized chapter bibliographies which helpfully include authors' affiliations and references to published reviews of cited articles.

Several areas exist where the author's judgment could legitimately be questioned (such as the failure to even mention FORTRAN in the body of the chapter on automatic programming) and, in rare instances, he is directly in error (as when he states that the minimum form of an incompletely specified sequential machine can always be found by considering all possible combinations of values for the *don't care* entries.[1]) A larger failing is the fact that the method of presentation depends almost entirely upon the written word, and illustrations are sadly scarce. Less than 5 percent of the pages contain a figure or table of any type and, for example, verbal descriptions are relied upon exclusively when discussing the operation of stepping or cross-bar switches.

The basic contributions of the book nevertheless remain, and it is recommended to those with some prior experience in digital systems who wish to gain a good engineering understanding of the indicated areas.

CHARLES V. FREIMAN
IBM Corporation
Menlo Park, Calif.

[1] M. C. Paull and S. H. Unger, "Minimizing the number of states in incompletely specified sequential switching functions," *IRE Trans. Electronic Computers*, vol. EC-8, pp. 356–367, September 1959.

R67-52 Adaptive Systems of Logic Network and Binary Memories—J. Aleksander (*1967 Spring Joint Computer Conf., AFIPS Proc.*, vol. 30. Washington, D.C.: Spartan, 1967).

This paper presents a conception, and suggests physical implementations, of a completely general logic system having N binary inputs and M binary outputs, which can be controlled externally by $M2^N$ control-input wires to realize all 2^{M2^N} logic functions. Several applications are suggested, as a "trainable" logic system, as a function generator (A, log A, etc.), and as a pattern classifier. This paper is interesting and clear, although it has many typographical errors.

The author of this paper presents his system as an alternative to adaptive threshold nets in equivalent applications. It is useful to compare the performance of the proposed system with the performance of certain forms of threshold networks with regard to speed, amounts of equipment required, and ability to extrapolate or to "generalize."

The author shows first that the system can be broken down to M separate modules; each module provides one of the required M binary outputs. In training a single module, one method tries all possible logic functions and requires up to 2^{2^N} operations. A better search method is presented which only requires up to 2^N operations. The author points out that when using a 10-MHz clock, a module with 6 inputs would require about 1.6×10^{12} seconds or about 50 000 years to search using the first method, but this would be reduced to 6.4 microseconds using the second method. Although the author's example is a dramatic one, it must be realized that even the second method for "adapting" or setting the logic module is very slow when the number of inputs is increased. For example, when there are 100 inputs, up to 2^{100} operations are required to train the module. With a 10-MHz clock, this would take about $1.3(20)^{23}$ seconds or about $4(10)^{12}$ years.

The training time of the proposed system increases exponentially with the number of inputs. On the other hand, the training time of threshold elements and of simple parallel threshold networks increases approximately linearly with the number of inputs.[1,2] A "rule of thumb" found from much experimentation and experience is that the number of iterations required to train a threshold element is of the order of five to ten times the number of inputs when the number of training patterns is large (close to but less than the "statistical capacity"[2]). Accordingly, a digital threshold element with 100 inputs capable of 1000 iterations or adaptations per second could adapt to a desired logic function (or perhaps only come close to such a function if the function is nonseparable) in about one second. If a simple

[1] B. Widrow, "Generalization and information storage in networks of Adaline 'neurons'," in *Self-Organizing Systems 1962*, M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, Eds., Washington, D. C.: Spartan, 1962.

[2] K. Steinbuch and B. Widrow, "A critical comparison of two kinds of adaptive classification networks," *IEEE Trans. Electronic Computers*, (*Short Notes*) vol. EC-14, pp. 737–740, October 1965.

parallel net of the "Madaline"[1,2] type is used, more complicated piecewise hyperplanar separating boundaries are realizable. The training time increases approximately in proportion to the number of hyperplanes, as has been established experimentally. A ten-threshold-element digital system with 100 binary inputs would adapt in about 10 seconds.

The amount of equipment required in the realization of threshold networks is proportional to the number of outputs and to the number of inputs. On the other hand, the amount of equipment needed to implement the proposed systems increases in proportion to the number of outputs, but increases exponentially to the number of inputs. One scheme proposed by the author would require a single output OR element and $2^{100} \approx 1.3(10)^{30}$ AND elements for one 100-input module. In addition, 2^{100} input control circuits would be required. Although threshold elements are more difficult to build than conventional logic elements, the amount of equipment required by threshold elements is far less when there are many input circuits and when the desired logic function need not be realized perfectly precisely (as is the case in statistical pattern classification systems). The training time of threshold-element systems is relatively far less when there are many inputs.

The author describes several realizations of his system using solid-state logic elements. In Widrow,[1] it has been shown how almost the same system can be realized using a conventional magnetic-core memory. An N input M output system would require a core stock containing M planes, each being $2^{N/2} \times 2^{N/2}$. For example, a 16 input system would have an 8-bit X address and an 8-bit Y address. Each memory plane would therefore be 256×256. It is again apparent that a large amount of equipment would be needed even when the number of inputs is moderate. Each binary input vector would correspond to an address. The responses required on the M output lines would be stored in the corresponding memory register. The core-memory realization has the advantage over the author's realizations in that it provides the means for storing the desired logic function in addition to simply providing for externally adjustable logic. The author's realizations have the advantages of being faster in responding. The core memory requires a fraction of a memory cycle to respond to an input vector.

It is interesting to compare the performance of the author's system with that of threshold nets from the point of view of generalization. By generalization, the following is meant. A number of entries of the truth table are given, called training samples, from which the remaining entries in the truth table must be inferred. The problem is not unlike that of interpolating a function when given only a finite number of samples of that function. In order to do this, something about the function must be known, such as the function is band-limited, or the function is polynomial of a certain degree. The number of samples must be adequate to represent the function. To interpolate the remainder of the responses of a truth table given a set of samples, something must be known about the truth table. For example, it might be known that the truth table is linearly separable, or separable with a finite number of hyperplanes, or separable with a quadratic boundary, etc. With such knowledge, a suitably restricted form of adaptive logic structure could interpolate the entire truth table by adapting to fit an adequate number of training samples. A perfectly general adaptive logic system has no inherent ability to generalize. Being perfectly general, it could fit every sample of every truth table. Therefore, every sample of a desired truth table must be given, and no generalization is possible.

Although the author's system is perfectly general and, therefore, not inherently capable of generalization, the author shows how it can be used to generalize. By restricting the interrelations among the control input signals, the author shows how a hyperplanar separating boundary can be realized and how the controls can be changed to cause rotation of this hyperplane. That nonlinear boundaries can be realized by other control functions is stated, and that it is possible to translate and change the shapes of such boundaries is clear. However, the way in which the properties of separating surfaces are related to the control functions and the way in which control functions are chosen to satisfy real problems are not discussed. Just as things become very interesting, the paper ends.

To summarize, the general scheme proposed in this paper could be used when the number of inputs is small and the logic-function requirements are very general. On the other hand, threshold logic elements could be used (either analog or digital units) when the nature of the application requires such functions, especially when the number of inputs is large. Examples of such applications are given in Koford and Groner[3] (statistical examples) and in Smith[4,5] (deterministic examples). When the number of inputs is large and the logic-function requirements are very general, no satisfactory solution is in sight.

BERNARD WIDROW
Stanford University
Stanford, Calif.

[3] J. S. Koford and G. F. Groner, "The use of an adaptive threshold element to design a linear optimal pattern classifier," *IEEE Trans. Information Theory*, vol. IT-12, pp. 42–50, January 1966.
[4] F. W. Smith, "Design of quasi-optimal minimum-time controllers," *IEEE Trans. Automatic Control*, vol. AC-11, pp. 71–77, January 1966.
[5] F. W. Smith, "A trainable nonlinear function generator," *IEEE Trans. Automatic Control*, vol. AC-11, pp. 212–218, April 1966.

B. LANGUAGES

R67-53 SIMULA—An ALGOL-Based Simulation Language—Ole-Juhan Dahl and Kristen Nygaard (*Commun. ACM*, vol. 9, pp. 671–678, September 1966).

The authors of this article are primarily interested in describing a new language for programming discrete event simulations. In actuality, the language described has much wider use than simulation, and may have effects that are felt outside the simulation community. SIMULA is not revolutionary in concept; it contains few innovations. Its strength stems from the fact that its authors have relied heavily on the work of their predecessors and have selected technically sound ideas for creating an advanced programming language. In essence, SIMULA is a successor to ALGOL, containing all of its constructs together with facilities for list processing and parallel sequencing of control.

List processing is a natural extension of high-level algorithmic languages, and is especially natural to accommodate the dynamic data structures characteristic of simulations. SIMULA borrows its facilities from SLIP,[1] particularly making use of ring-organized lists and reference counters to implement storage reclamation. The flexibility and power obtained from SLIP-like data structures give SIMULA a definite advantage over SOL,[2] another ALGOL-based simulation language with a much less flexible data structure. Because of its list-processing facilities, SIMULA is an attractive language to use for non-simulation programs.

The SIMULA programming system makes use of its list-processing capabilities to control the parallel execution of stimulated processes. Parallelism is described through the equivalent of the "fork" construct, which conceptually causes control to transfer to a remote process as well as proceed in sequence. Actual processing is purely sequential and is determined by scanning a master-timing list, called the sequencing set, that contains a schedule of events. Parallel control statements are implemented as list-processes on the sequencing set. It is well to note here that sequential execution is a property of the

[1] J. Weizenbaum, "Symmetric list processor," *Commun. ACM*, vol. 6, pp. 524–544, September 1963.
[2] D. E. Knuth and J. L. McNeley, "SOL-A symbolic language for general purpose systems simulation," *IEEE Trans. Electronic Computers*, vol. EC-13, pp. 401–408, August 1964.