therefore nearly open-circuited. The voltage rate of change $dV_L(t)/dt$ at the output is therefore nearly twice the voltage rate of change $dV_i(t)/dt$ at the source end [6]. This effect shortens the effective delay in the line (Fig. 1) by at least $t_r/4$ [3], where t_r is the rise or fall time of V_i measured between 10 and 90 percent of its amplitude [3] at the input of the distortionless transmission line.

References

[1] L. S. Garret, "Integrated-circuit digital logic families, Part 3—ECL and MOS devices," *IEEE Spectrum*, vol. 7, pp. 30–42, Dec. 1970.

[2] A. Nguyen-huu, "An analysis of the ringing phenomenon in digital system," *Comput. Design*, vol. 10, pp. 39–45, July 1971.

[3] "MECL 10,000 series: Schottky diode termination," Motorola Semiconductor Products, Inc., Feb. 1971.

[4] O. A. Horna, "Pulse reflection in transmission lines," *IEEE Trans. Comput.* (Short Notes), vol. C-20, pp. 1558–1563, Dec. 1971.

[5] "Solid state circuit devices," in *Hewlett-Packard Catalog 1969*, pp. 19–23.

[6] J. L. Potter and S. J. Fich, *The Theory of Networks and Lines*. Englewood Cliffs, N. J.: Prentice-Hall, 1963, pp. 401–403.

An Automatic Test Generation System for Illiac IV Logic Boards

VISHWANI D. AGRAWAL, MEMBER, IEEE, AND PRATHIMA AGRAWAL

Abstract—A test generation system, developed for the logic boards of the Illiac IV computer, is described. The system combines the test generation by random patterns and the *D*-algorithm. Some results are given to illustrate the effectiveness of this approach.

Index Terms—*D*-algorithm, fault diagnosis, off-line computer diagnosis, random test generation, test generation.

Introduction

The Illiac IV computer [1] uses a very large number of electronic components distributed over printed circuit boards in the form of integrated circuit packages. Each processing element (there are 64 of these in the computer) consists of about 180 logic boards, each containing up to about 100 gates. Control unit boards are about four to eight times larger and use multilayer printed circuitry. All of these boards are tested by a separate diagnostic computer which either must be supplied with the previously generated tests or should be able to generate the tests from the circuit description. This note describes a system developed to generate these tests. As usual, only the stuck at "0" and stuck at "1" type faults are considered. The system was designed to produce, with least human intervention, a complete set of input patterns for detecting all the faults in a board, under the single fault assumption. This diagnostic system was programmed in Fortran IV and uses as its input the circuit description obtained from the design automation files of the Illiac IV boards. The results reported here were obtained on a Burroughs 6500 computer.

Test Generation by Random Inputs

Ideally, a logic network of n primary inputs can be completely checked by observing its response to 2^n possible input patterns. However, if after each input pattern, a record of all the faults detectable by this pattern is kept, the testing can be stopped as soon as all the faults have been accounted for. This procedure often results in much less than 2^n test patterns. A complete test generation can be guaranteed only if the system is capable of using all the 2^n inputs. For large circuits, this becomes impractical unless the circuit can be partitioned into independent sections. This partitioning, often impossible, has to be done manually and therefore, is not considered here. First we will determine how much of test generation can be achieved by using a practical number of input patterns. In general, the input patterns can be obtained randomly and the detectable faults determined by using the logic simulators of the fault free and the faulty networks, successively [2]. If storage elements are present, their states are initialized by an auxiliary program. Even when single faults are considered, this procedure results in a very large number of simulations. Considerable effort can be saved by using the TEST–DETECT [3] program, as described below.

The flow chart of the test generation system is shown in Fig. 1. The circuit description data consist of the information about each line. The lines are defined as either primary inputs or the outputs of gates and are numbered by integers 1, 2, $\cdots$, starting from the primary inputs. The fault list simply consists of the stuck at "0" and stuck at "1" faults at all the lines. In this section we assume that the next block in the flow chart always makes the choice (1). This corresponds to the test generation entirely by random patterns. For each input line, a random floating-point number x, uniformly distributed in the interval [0, 1] is generated. If x is less than 0.5, the line is set to "0," otherwise, it is set to "1." Next, for these inputs, the values of all the remaining lines are determined by successive use of forward implications [3]. In forward implication, the output of a gate is determined as soon as enough inputs are known, e.g., a single 0 at any of the inputs of an AND gate establishes its output to be 0. The TEST–DETECT program is then used to give a list of all the faults detectable by the input pattern under consideration, the primary output lines at which each fault can be observed and the change in the output value caused by the fault. For details of TEST–DETECT, one is referred to [3]. Now this list of detectable faults is compared with the fault list of the circuit and the common faults are eliminated from the fault list. This process is repeated with a new random input every time until each fault has been detected at

Manuscript received December 20, 1971; revised March 17, 1972. This work was performed under the Illiac IV contract.

The authors were with Automation Technology, Inc., Champaign, Ill. V. D. Agrawal is now with EG&G, Inc., Albuquerque, N. Mex. 87106.

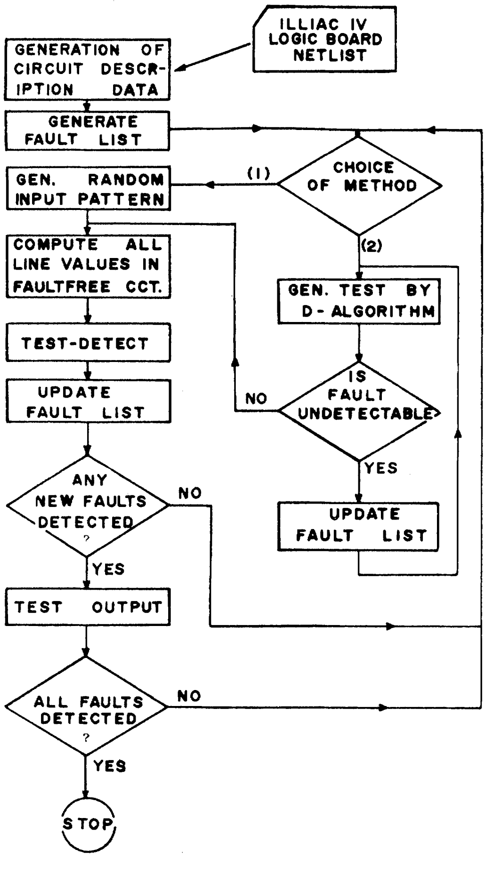

Fig. 1. Flow chart of test generation.

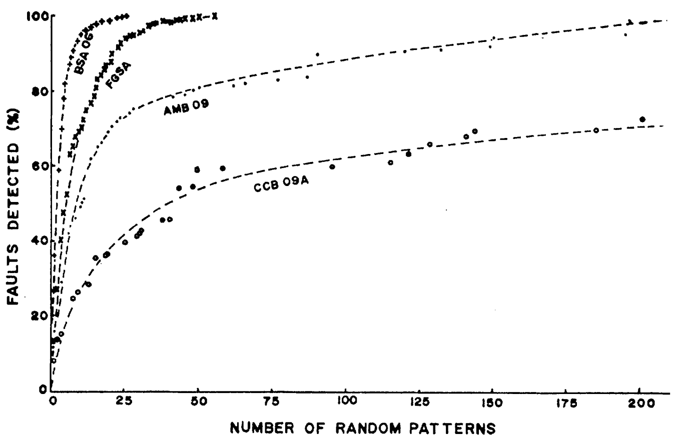

Fig. 2. Results of random test generation.

least by one pattern. Any pattern, that detects at least one new fault, undetected by all the previous patterns, is stored as a test.

Results of this program on some of the boards are shown in Fig. 2, which indicates the percentage of faults detected as the number of input patterns increases. The points in this figure correspond to the test patterns and the curves show the trend of test generation. Logic boards are identified by their Illiac IV names and some of their details may be found in Table I. Initially, the test patterns are generated at a fast rate for all boards, however, for some boards (e.g., CCB09A), the process soon becomes extremely inefficient. When the execution time of the program was limited to 15 min, several boards were left unfinished. In Table I, it appears that the deciding parameter for the completion of test generation is the number of levels in the board. Number of levels is determined from the longest chain of gates between a primary input and a primary output. Primary input is counted as level 1 and every time a gate is encountered along the chain, the level is incremented by 1. Thus the level of the output is the largest level in the board. Whenever the number of levels exceeds 8 or 9, the test generation is left incomplete. These results indicate the need for another, preferably deterministic, method.

TABLE I

COMPARISON OF TEST GENERATION SYSTEMS

Illiac IV Board Name	Number of Lines	Number of Levels	Time of Test Generation (s)		
			Random Method	*D*-Algorithm	Combination
BSA06	124	5	38	67	38
SGA03	112	6	63	88	64
FGSA	437	6	250	544	250
AMB09	89	7	79	65	49
CSA04	189	8	incomplete	216[a]	94[a]
CLA11	128	9	incomplete	125	114
CCB09A	115	10	incomplete	137	146

[a] One fault found undetectable due to redundancy.

APPLICATION OF *D*-ALGORITHM

For generating a test for a given fault, the *D*-algorithm [3] has been found to be ideally suitable. It always generates a test whenever one exists. Recently, its application has been extended to sequential circuits [4]. A *D*-algorithm test generation program was written in Fortran IV. This program, when supplied with the circuit description and a fault list, either finds a test for the first fault in the list or declares this fault undetectable. If the test is found, any remaining DON'T CARE inputs are then determined randomly and TEST–DETECT is applied as before. In the flow chart of Fig. 1, if choice (2) is always made, we get a test generating system using only the *D*-algorithm. This system generated complete tests for all the boards (Table I). For the board CSA04, one fault reported as undetectable was in fact found to be due to redundancy in the logic. This system appears to be slower than the random pattern method for the boards with six or less levels. This is perhaps due to the comprehensive treatment of the fault under consideration in the *D*-algorithm.

The above results suggest that a system designed to handle boards of various sizes, should combine the two methods. It was found that the execution time of one complete cycle in Fig. 1 with choice (1) was approximately one-fourth that with choice (2). Therefore, the following strategy was adopted. The program is started with choice (1) and continued like this until it is found that four consecutive random patterns failed to detect any fresh fault. Thereafter, the choice is changed to (2). This system produced significantly improved results in

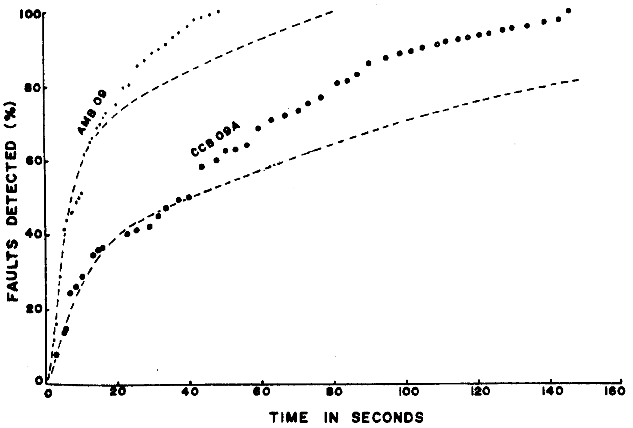

Fig. 3. Results of test generation by combination method.

almost all cases (Table I). For six or less levels, the time of test generation was the same as that required by random patterns alone since the test generation was complete even before the program switched to D-algorithm. For seven or more levels, where random patterns were very inefficient, the combination method not only completed the test generation but also resulted in faster speed. The results of this system on two boards are shown in Fig. 3, where the test patterns are indicated by points against their generation time. For comparison, the trend of entirely random test generation is also shown by dashed lines. These results are representative of the results obtained on a large number of logic boards.

Conclusion

The application of the two test generation methods to the Illiac IV logic boards has shown that: 1) for smaller number of levels (typically, less than seven), the random patterns provide an efficient method of test generation; 2) for more than seven levels, D-algorithm is necessary to complete the test generation; and 3) redundancies, if present, must be accounted for by D-algorithm. As the results in Table I show, the combination of the two methods appears to be an optimized system. This new system also eliminates the need for human intervention as it takes care of the boards of various sizes automatically.

The random pattern test generator described in this note uses the TEST–DETECT program instead of the more conventional logic simulators [2]. All the results reported here were obtained for combinational logic. However, it is believed that a similar system can be devised for circuits containing sequential logic since both the methods used here have been extended to sequential circuits [2], [4].

References

[1] D. L. Slotnick, "The fastest computer," *Sci. Amer.*, vol. 224, pp. 76–87, Feb. 1971.

[2] V. Moreno, "A logic test generation system using a parallel simulator," Dep. Comput. Sci., Univ. Illinois, Urbana, Illiac IV Document 243, Feb. 1971.

[3] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed algorithm to compute tests to detect and distinguish between failures in logic circuits," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 567–580, Oct. 1967.

[4] W. G. Bouricius, E. P. Hsieh, G. R. Putzolu, J. P. Roth, P. R. Schneider, and C.-J. Tan, "Algorithms for detection of faults in logic circuits," *IEEE Trans. Comput.*, vol. C-20, pp. 1258–1264, Nov. 1971.

High-Speed Multiplication Systems

A. A. KAMAL, MEMBER, IEEE, AND M. A. N. GHANNAM

***Abstract*—Large-scale scientific computers are characterized by having a high ratio of computing requirements to the input–output requirements. Thus the speed of arithmetic operations should be of prime importance in this group of digital computers. In this note a mathematical formula is derived to determine the multiplication time obtained by each of the known high-speed multiplication techniques. Then a comparative analysis is performed to show the amount of speed improvement resulting from each technique and the influence of the different factors affecting speed.**

***Index Terms*—High-speed multiplication, multiplier addition logic, multiplier ternary decoding, multiplier variable shift decoding, registers connection logic.**

I. Introduction

The system design of an arithmetic unit has different aims that vary in importance according to design and applications. These goals are speed, cost, complexity, and reliability. With large-scale scientific computers the problems handled are characterized by having a high ratio of computing requirements to the input–output requirements. Thus speed should be of prime importance in this group of digital computers. This work provides a comparative analysis for the behavior of the different speeding-up techniques used with binary multiplication. This helps to identify the amount of improvement in speed attainable with each technique that is a prerequisite for applying different criteria of selection such as complexity, economy, and reliability. Starting with the logical steps required to perform multiplication, a mathematical formula is derived to determine the multiplication time obtained by each technique following the basic assumptions given by Flores [1]. These formulas are then used to calculate the amount of improvement as a function of the various factors affecting speed.

II. Binary Multiplication

Binary multiplication systems can be classified according to the addition logic used, namely: 1) multiplication systems using carry propagate adder (CPA) [1]; 2) multiplication systems using carry propagate/carry save (CP/CS) adder [1]; 3) multiplication systems using cascaded carry save adders (CCSA) [2]; and 4) multiplication systems using adder tree (AT) [3]–[5].

The constant-shift multiplier-decoding schemes (ter-

Manuscript received September 27, 1971; revised March 15, 1972.

A. A. Kamal is with the Department of Electronic Engineering, Cairo University, Giza, Cairo, Egypt.

M. A. N. Ghannam is with the Electrical and Electronic Engineering Laboratory, National Research Center, Giza, Cairo, Egypt.