THE BELL SYSTEM
TECHNICAL JOURNAL

DEVOTED TO THE SCIENTIFIC AND ENGINEERING
ASPECTS OF ELECTRICAL COMMUNICATION

Volume 53 October 1974 Number 8

LAMP:

System Description

By H. Y. CHANG, G. W. SMITH, Jr., and R. B. WALFORD

(Manuscript received February 28, 1974)

A general description of the Logic Analyzer for Maintenance Planning (LAMP) system is presented. LAMP is a digital-logic simulation and analysis system used for logic-design verification, for generation and evaluation of fault-detection and diagnostic tests, and for generation of the trouble-location manual (or fault dictionary) data. It is implemented on the IBM 360/370 TSS and OS machines (for both interactive and batch operations), and has been in active use at Bell Laboratories in the development of electronic switching systems, data set facilities, transmission equipment, and advanced integrated circuit technologies.

I. INTRODUCTION

The explosive evolution of digital devices, computers, and systems since the invention of the transistor has necessitated a parallel industry-wide development of tools for the design and test of logic circuits. Whereas the oscilloscope was the mainstay of the digital circuit designer in the early days of discrete-transistor logic circuits, it soon proved to be inadequate for design verification and fault-behavior testing of large systems employing integrated, digital logic. In response to this need for better logic-circuit-development tools, a host of digital-simulator algorithms and simulator systems has been produced.[1–3]

The need for reliable and dependable electronic switching systems (ESS) poses critical design problems. Computer-aided techniques can be used effectively for:

(*i*) Analysis and enhancement of system diagnosability.
(*ii*) Logic-design verification.
(*iii*) Generation of fault-detection tests.
(*iv*) Analysis of faulty-circuit behavior.
(*v*) Verification and evaluation of diagnostic-test designs.
(*vi*) Production of trouble-location manuals (TLMs).

The LAMP system has been designed to attack these problems in a systematic manner.

This paper provides a brief description of the LAMP system organization and features, and is intended to serve as background for the four following papers. These provide details of the logic simulators, the automatic-test-generation system, and the techniques for organizing system design for diagnosability.[4–6] They include a specific example of how LAMP was employed in the development of a large processor for an electronic switching system.[7]

II. EVOLUTION OF THE LAMP SYSTEM

The decision to build a machine-aids system with digital-simulation capability was motivated by the successful use by Bell Laboratories designers of the sequential analyzer.[8] The use of this simulator showed the great advantages of using simulation for logic testing and fault diagnosis. By 1966, Bell Laboratories was incorporating simulation techniques into the design cycle of electronic-switching-system equipment. However, there were several difficulties in the day-to-day use of this simulator. It had a restrictive logic model, long turnaround time due to remote computer location, and no capability for handling large circuits (for example, circuits having as many as 10,000 gates). Because no simulator then available could meet the growing demand for logic-simulation service, a decision was made to develop an advanced logic-simulator system which would grow and adapt to Bell Laboratories current and future needs.

It is instructive that the motivation to develop a design-aids system came from the potential users of that system. Likewise, the initial design objectives and the evolution of the system were influenced to a large extent by the intended users. This has resulted in a very sophisticated, user-oriented system which continues to grow and evolve to meet the changing requirements of the designer.

The initial system was made available to users in late 1969 on IBM System/360 TSS at Bell Laboratories, Naperville, Illinois. It had only a modest set of features. However, the user reactions were generally favorable. Since then, substantial improvements in system performance and capabilities have been incorporated. The TSS version of LAMP was converted to run on IBM System/360 OS in mid-1970 and was made available to Bell Laboratories users at Holmdel, New Jersey, and Columbus, Ohio. Automatic-test-generation capability was incorporated in early 1972; and the facilities to analyze system structural diagnosability were implemented in late 1972. The LAMP system is in active use in the development of many ESS projects as well as other non-ESS work such as the development of data-set facilities, transmission equipment, and advanced integrated-circuit technologies. The current user group includes twenty organizations from nine Bell Laboratories locations (Murray Hill, Whippany, Holmdel, Allentown, Columbus, Merrimack Valley, Indianapolis, Denver, and Naperville).

III. SYSTEM ORGANIZATION

LAMP is a system of programs designed to be used for logic-design verification, evaluation of fault-detection tests and diagnostic programs, and generation of the trouble-location manual (or fault dictionary) data. It is implemented on the IBM 360/370 TSS and OS machines (for both interactive and batch operations). The current version can handle circuits containing up to 65,000 gates. The system is composed of four basic parts:

(*i*) A circuit-description-language compiler.
(*ii*) A command-language interpreter.
(*iii*) A collection of design tools composed of an automatic-test-generation (ATG) system;[4] a controllability, observability, and maintenance engineering techniques (COMET[5]) system; and a family of simulators.[6]
(*iv*) An output system.

A block diagram showing the functional relationship of the various parts of the LAMP system is presented in Fig. 1. A logic circuit can be described to the LAMP system through a user-oriented language called *LSL-LOCAL*. The circuit description is then translated by the language compiler into *simulation tables*. The *command-language interpreter* directs all the actions of simulation, test generation, and diagnosability analysis in accordance with user-specified commands and information stored in the simulation table.

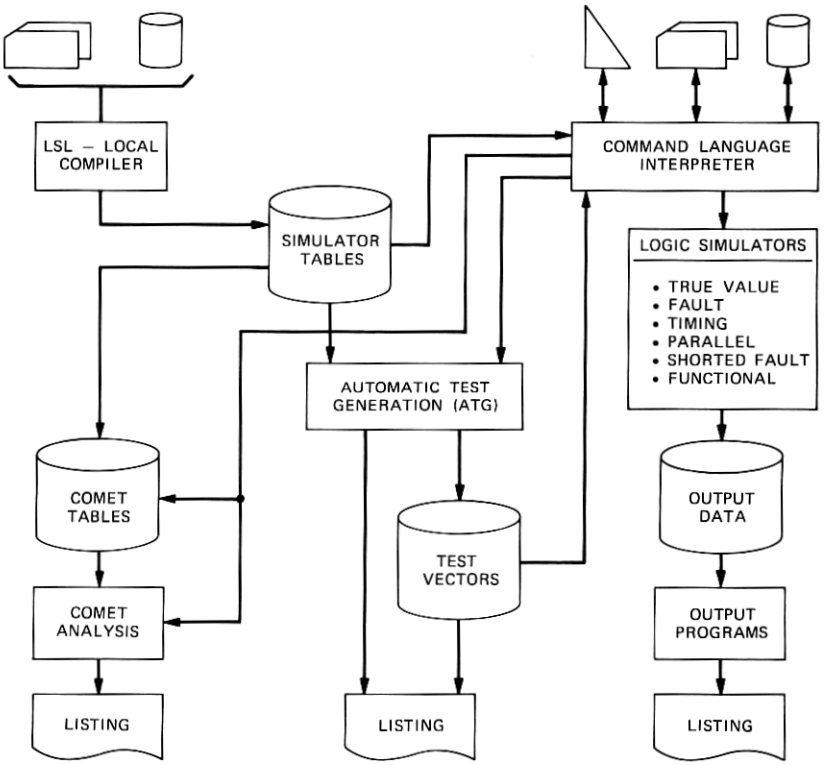

Fig. 1—Block diagram of LAMP system.

For a given logic-circuit description, the ATG system can automatically produce the test-vector information. To verify logic design and to study faulty-circuit behavior, a family of *simulators* can be used. The inputs applied to the simulators can be manually generated and/or generated by the ATG programs. The simulators are capable of simulating circuit behavior in either fault-free or faulty mode, with facilities to handle race and oscillation conditions and to perform detailed timing analysis.

If the purpose at hand is to determine the diagnosability of the design, the *COMET system* can be used to assist the users to organize systems design for diagnosability by systematically determining the optimum placement of control-access and monitor points. Simulation and analysis results are then collected under the control of an *output system*. Numerous output options can be specified that allow users to obtain information concerning logic verification, timing analysis,

and other data-processing information at the time of simulation or afterwards.

In the following sections, the salient features of the various major functional blocks in the LAMP system will be described.

3.1 Circuit description input language

A logic circuit is described to the LAMP system through a user-oriented language called LSL-LOCAL. This language permits the entry of all information concerning the particular circuit either at the gate level or at the functional level. At the gate level, circuits are described in terms of logic elements such as NANDS, NORS, ANDS, ORS, and NOTS, whereas the functional level the circuits are expressed as memories, registers, clocks, etc. LSL-LOCAL was designed as an easily extendible language, to be used by circuit designers and diagnosticians who may not be trained as programmers.

Once the circuit description is entered, the LSL-LOCAL language processor compiles the description into data tables to be used by the simulator(s), the ATG system, and the COMET analysis programs. The language processor has a substantial number of checks built into it to detect and intercept most errors before they can get into the system. These checks include syntax checks (for missing parameters, illegal characters, etc.) and circuit connectivity and consistency checks such as fan-in/fan-out limits. These features enable the users to check the coding of a circuit efficiently in terms of cost and time.

The original version of the language processor was developed in late 1969. Since then, three major revisions have been implemented to enhance its capability and performance. Many of the improvements were incorporated to support a wider range of applications, and the language has become a standard logic design input language in Bell Laboratories.

As an example of the LSL-LOCAL circuit description, an exclusive-OR circuit as shown in Fig. 2a can be encoded as:

```
CKTNAME:  XOR;
  INPUTS:  A, B;
 OUTPUTS:  X;
     NOT:  A′, A;
           B′, B;
    NAND:    AB′,        (A, B′);
             BA′,        (B, A′);
             A × B,      (AB′, BA′),    (X);
           (gate name)  (input list)  (output)
```

The description generally consists of three parts: (*i*) the CKTNAME statement, which introduces the circuit description and declares the name of the circuit; (*ii*) connection declarations, which specify the names and the types of all the input/output signals of the circuit; and (*iii*) interconnection blocks, which specify elements and networks used in the circuit and how these are interconnected. The hierarchical structure of the language allows the specification of circuits in a modular fashion. Thus, the exclusive-OR circuit can be used as an element in describing a single-bit adder [see Fig. 2(b)]:

```
CKTNAME:  ADDER1;
  INPUTS:  A, B, K;
 OUTPUTS:  C, K_;
     XOR:  A × B, (A, B), (X);
           D, (X, K), (C);
     NOT:  A′, A;
           B′, B;
    NAND:  ANB, (A, B);
           AORB, (A′, B′);
           AORBNK, (AORB, K);
           K_, (ANB, AORBNK);
```

These single-bit circuits can then be used to describe an n-bit adder or other more complex logic element(s). There is no explicit limit on the number of levels of nesting in describing circuits using LSL-LOCAL. A user can very conveniently construct the data base of a large circuit or system by combining the various data bases from its component circuit modules.

3.2 Command system

The control of LAMP system action for test generation, simulation, and COMET analysis is accomplished by means of a command-language structure. A large selection of interactive commands is available which enables the users to compile and edit a circuit description, specify simulation-test vectors, make simulation runs, observe circuit behavior, gather circuit statistics, determine optimal placement of maintenance-access and observation points to enhance diagnosability, and specify types of simulation and analysis output. At present, there are approximately 80 commands in the system, many of which were implemented at the request of users. The commands are highly user-oriented so that one can easily learn the use of the system after a relatively minor amount of instruction.

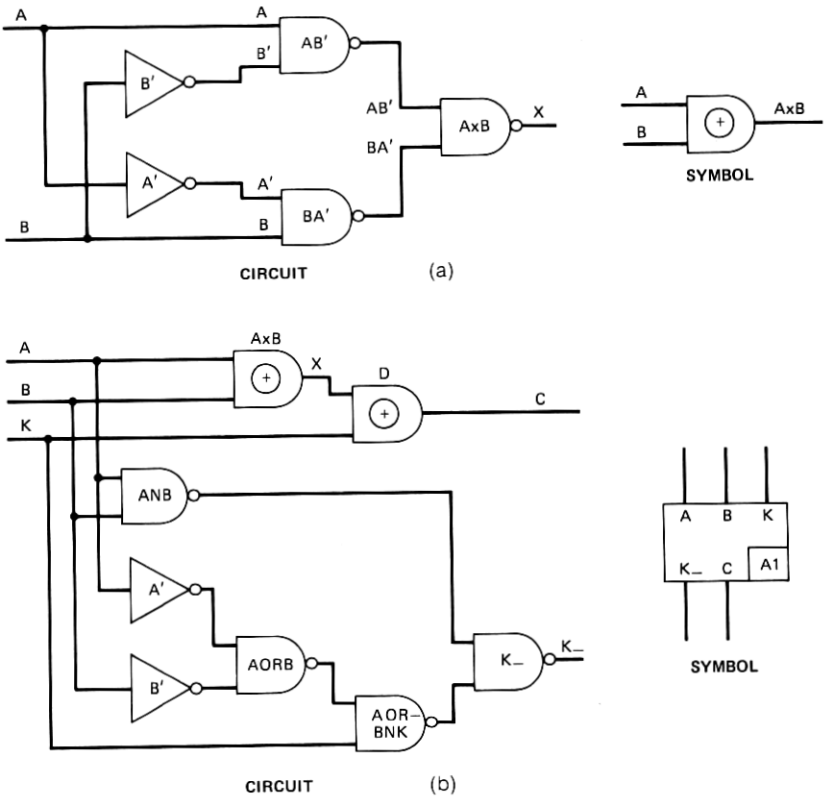

Fig. 2—(a) Exclusive-OR circuit. (b) One-bit adder circuit.

The system structure is implemented with four levels of hierarchy. On the base level is the executive routine which reads commands entered by the system user and interprets them as to type. It then calls the appropriate routine to handle the command. On the next level are the command handlers whose functions are to process the command line and call the appropriate functional processors and service routines. On the third level are the functional processors; they are designed to perform specific functions such as simulation, circuit-description and test-vector compilation, circuit modification, processing control, and interactive control. On the fourth level are the various service routines that perform such tasks as gate-name retrieval, print control, vector translation, preliminary processing of data lines, file accessing, and printing.

To illustrate the richness of the command language, a few of the most commonly used commands for logic simulation are described. Referring to Fig. 3, to enter circuit descriptions into LAMP, the LSL-LOCAL encoding of the circuit will be first compiled (using SOURCE) and the resultant simulator tables loaded (using LOAD). A circuit can also be formed by combining several circuits into one using LINK. Should it become necessary to modify the circuit logic without recompiling the entire circuit, then CKTCHANGE can be used to connect/disconnect gates, add gates, and rename, change, or remove gates.

The input test vectors for simulation can be described in either trinary (0, 1, and "don't know"), octal, or hexadecimal form (using INVEC), or in terms of vector names defined by PATTERN. In certain applications, the STATE command is used to set the circuit-state variables to initialize a circuit before a simulation run. Internal gates of the circuit can be treated as additional circuit outputs or test points by issuing the MONITOR command. Conversely, normal circuit-output leads can be MASKED out for a particular run.

The what, when, and how much of the simulation statistics that are to be processed after a run are defined through RESULTS. A simulation is initiated by the RUN command and can be temporarily halted by a STOP command. At a STOP, the user may interrogate the state of the simulation and obtain simulation statistics accumulated up to that point (by using the DISPLAY command), or he may overwrite the next input vector in the INVEC data set by issuing an ALTER command. The simulation can be resumed by issuing a GO command. If the user wishes to change the course of simulation during a STOP, he can use the JUMP feature to skip unwanted test vectors.

To facilitate the use of the LAMP system in the production mode, many commands have been developed for analyzing circuit topology, gathering circuit statistics, and performing audits. Some examples are the CKTCHECK command to check the consistency of simulation tables and to provide statistical information such as counts of gate and functional types, average fan-in and fan-out for each type, percentages of types to total, etc., and the CKTSTAT command which prints a brief summary of circuit statistics including number of gates, number of circuit inputs, number of circuit outputs, number of clocks, and number of nonfaulted gates. For topological analysis, the LOOPS command allows one to identify all loops within a circuit or contained by a specified gate, the FEEDBACK command identifies the minimum number of feedback loops within the circuit, the PATH command finds the shortest path between a specified gate and any input, and the MSC

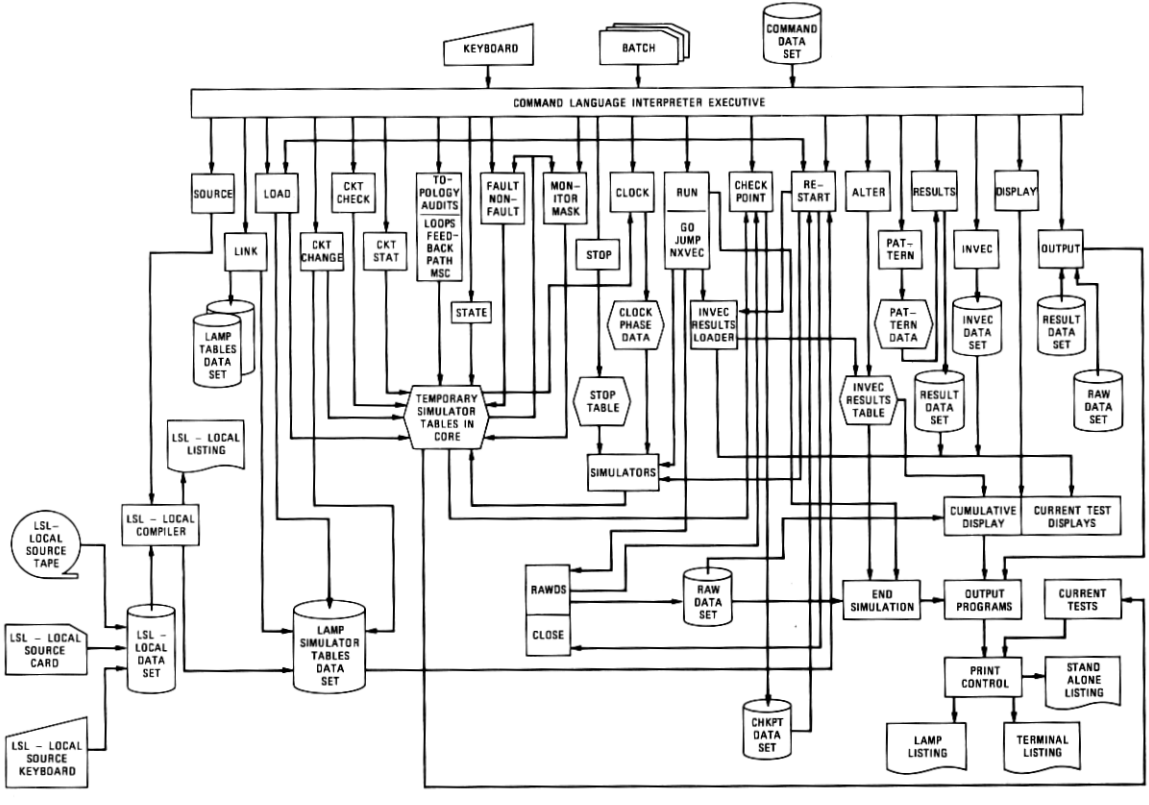

Fig. 3—Examples of commands used in simulation.

command identifies all maximally strongly connected sets of gates within the circuit. All these commands have been proven to be extremely useful, especially in the course of simulating large circuits (e.g., those containing 50,000 gates) under fault conditions.[7]

While the LAMP commands generally assume interactive use of the system (on 360/TSS), they also permit the use of the system in the noninteractive mode (such as 360/TSS batch or 360/370 OS). In these cases, some advance planning must be done to enable the runs to be completed successfully.

3.3 Major tools

There are three major tools in the LAMP system: an automatic-test-generation (ATG) system, a family of simulators, and a system for diagnosability analysis (COMET). Detailed descriptions of these tools are covered in the companion papers.[4–6] The purpose of this section is to describe the salient features of these systems and to briefly describe the interactions among them and the rest of the LAMP system.

3.3.1 Automatic-test-generation (ATG) system

ATG is a system of programs that can automatically produce the test-vector information for a given logic-circuit description. The faults considered are the classical input open, output stuck-at-one, and output stuck-at-zero for each gate in the circuit. There are two major differences between ATG and those test generators that have been discussed in the literature.[9,10] First, ATG is capable of handling both combinational and sequential circuits without the need to identify feedback lines. Second, the system treats logic circuits as an interconnection of unit- and zero-time-delay gates, and thus improves the accuracy of the circuit modeling.

ATG interacts with other parts of the LAMP system via the command-language interpreter (see Fig. 1). A set of about 20 commands is available to the user to set the initial conditions (e.g., loading the circuit description, specifying sequence length of the test), select test-generation strategies, specify output procedure, and direct the general course of action. The fault-detection level achieved by the tests generated by ATG can be evaluated by using the fault simulators available in the LAMP system. If the evaluation results indicate that the detection level is not adequate, ATG can be called again to generate more tests, by using different test strategies and/or changing the sequence length of the tests. This test-generation and evaluation loop

can be repeated several times until a specified level of detectability is achieved.

3.3.2 Controllability, Observability, and Maintenance Engineering Techniques (COMET)

Past experience has indicated that the effectiveness of diagnostic testing depends not merely on the techniques used in deriving tests and test results, but also on the inherent structural diagnosability of the unit.* The ATG system is a tool for aiding the derivation of test vectors for given circuits. The COMET system, on the other hand, employs a technique that enables one to determine for a given circuit the optimal placement of control-access and monitor points for diagnostic testing.

The COMET analysis is initiated by entering the connectivity of the functional blocks of a unit via LSL-LOCAL (see Fig. 1). The control and observation relations among the various functional nodes are automatically created from the connectivity (or simulator) tables. Through the use of the command-language interpreter, the user can then direct COMET to analyze, to examine, and to modify the topological structure of the unit. The modification of the structure for additional control and/or observation is performed automatically, or it can be explicitly directed by the user. Once the design has been COMETized, it enjoys the following advantages:

(*i*) Trouble-location-manual data can be generated and updated without the use of fault simulation.
(*ii*) Multiple faults and all nonclassical faults are locatable if they are detectable.
(*iii*) Diagnostic information can be easily updated in accordance with hardware change(s).
(*iv*) An orderly approach to the implementation of an overall diagnostic design is provided.
(*v*) The fault-location procedure is substantially simplified.

3.3.3 Logic simulators

In the heart of the LAMP system are the logic simulators. These are the programs that actually perform the simulation of the circuit under test. A total of six simulators is available, each of which is designed to

* Depending on the level of integration and the purpose at hand, a unit can be interpreted as a processor, a functional module, a circuit pack, or an LSI chip.

suit a particular condition.* Under the control of the command-language interpreter, one or more of the simulators can be called to simulate a particular circuit. The six simulators available in the LAMP system are:

(*i*) True-value simulator—This simulator simulates only the true-value (or nonfaulted) conditions of the circuit. Simulation is done at the gate level.

(*ii*) Fault simulator—This simulator can simulate the action of classical stuck-at-type faults (input open, output stuck-at-zero, and stuck-at-one) in addition to the true value. This enables one to study the behavior of faulty circuits, to evaluate the fault-detection capability of maintenance-check circuits and tests, and to generate diagnostic data for trouble-location-manual production.

(*iii*) Timing simulator—This simulator allows the specification of individual rise and fall times of all gates in the circuit but does not simulate the effect of the stuck-at faults. It is designed primarily for detailed timing analysis to verify that the circuit will work under worst-case conditions.

(*iv*) Parallel simulator—The features of this simulator are similar to the ones available in the fault simulator. The major difference is that the parallel simulator employs a technique whereby the true value and a small set of faults are simulated concurrently.

(*v*) Shorted-fault simulator—This simulator allows for simulation of nonclassical faults such as crossover shorts and shorts between adjacent paths. It is useful in aiding the design of manufacturing tests for circuit pack check-out.

(*vi*) Functional simulator—This simulator allows one to simulate the circuit behavior at a higher level (e.g., registers, memories, etc.) than at the gate level. Functional simulation is most useful in verifying initial logic design where detailed knowledge of gate-level logic is not available or the function(s) cannot be conveniently modeled by gate-level techniques.

The cost effectiveness of the LAMP system depends on the user's choosing the correct simulator or simulators for use in his application. Consequently, it was found necessary to combine the results of more

* This was found desirable and cost effective especially in a production environment where system performance and accuracy are often weighted against each other in the search for an optimum mix.

than one simulator if the model of one simulator is not sufficient for a particular application. This is accomplished by the output system.

3.4 Output system

In LAMP, a versatile output system is available that enables users to collect simulation and analysis results in one of several different formats (or in user-generated formats). Outputs may be specified at any time during or after the run. The results of several simulation runs may be combined together at some point after the simulation has taken place to produce the desired output. Simulation runs that are so combined may be from different simulators. All these options can be specified by the command language.

Among the various output options available, some of the most commonly used will be described here. To verify the validity of the logic design, the VALUES option is often used, which lists the inputs and outputs along with the (1, 0, and "don't know") values of outputs for a given input test vector. In some cases where one is interested in internal states of the circuit, one can use GATEIO option to display the value of selected gates and their driving and driven gates. This feature is especially useful during a simulation run when the run is temporarily halted or has gone into oscillation; another specific use of this feature is to display circuit connectivity. Another format often used to display the outputs of timing and the true-value simulators is TLGRAPH. TLGRAPH is an oscilloscope-like trace of the signals on the output gates, from the time the test is applied until the time the circuit settles down. Whenever the value of an output gate changes, the time interval is recorded as well as the output gate values. This format has proven to be extremely valuable in studying worst-case timing conditions.

A variety of output formats is also available for studying the completeness, accuracy, and resolution of diagnostic tests. The ATP format lists all the faults that have not been detected for the test sequence simulated. The RAW output format lists the output gate name, each gate's true value as well as the number of faults that causes each gate's true value to be complemented, and a listing of these faults. For a large run where a user is interested in only a summary of the run, the MATRIX output can be used to show the faults detected by each test; the result is presented in the form of a matrix or a fault table. If the user is interested in fault partitioning and diagnosability information, he can choose the TREE output that lists the test results in the form of a diagnostic tree by grouping all those faults causing the circuit to behave in the same manner for a particular test sequence.

Facilities are also provided to allow the user to write his own output processing program. The raw output data set (RAWDS) contains all the raw data on the output gates from a simulation, including information such as the input vector on each test for which raw data are collected, names of inputs and outputs, fault cross-referencing information, fault and nonfault information, and certain circuit statistics. The user can manipulate this information to create the desired output format. The availability of this feature has substantially reduced the burden that otherwise would be imposed on the LAMP system developers to meet the wide variety of user needs.

IV. THE ROLE OF LAMP IN THE DESIGN PROCESS

The process by which a logic design becomes a completed product has become very complex with the advent of integrated-circuit technology. This process is made even more difficult in the telephone industry because of the stringent up-time requirement of the switching machines.[11] The ability to diagnose any equipment failure thus becomes an important consideration in the design and implementation of these machines.

The design and implementation process for a new switching system processor is made feasible by the constant use of computer-aided-design tools. Figure 4 shows the overall implementation process from the initial logic designs through to the completed processor. It also illustrates how the various major features of the LAMP system can be used in each design step.

The start of any major logic design project is the specification of the system architecture along with the basic design decisions. The COMET feature of LAMP helps this process by providing information about the diagnosability of a proposed design. With this tool, the global diagnosability of a system design can be established. Once this overall design step has been completed, the logic can be partitioned into individual circuit packs and detailed circuit designs can begin. In this phase of the design, the designer uses the true-value simulator for design verification, and frequently uses the timing simulator to make sure that the logic-timing functions are correct.

The use of these simulators requires that the logic circuit be encoded in the LSL-LOCAL language. The encoding of the circuit in the LSL-LOCAL language at this point accomplishes two basic functions. The first function is to catch any circuit discrepancies through the use of audits in the language processor and the second is to provide a machine-readable form of the circuit design. This latter function is basic to the entire computer-aided-design function.

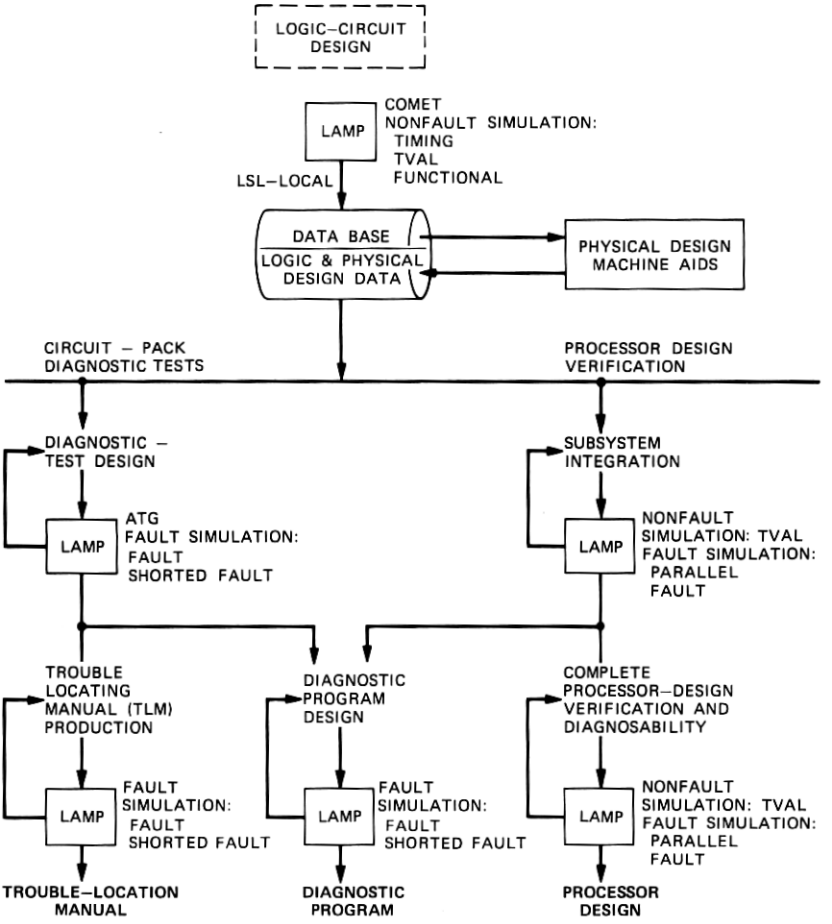

Fig. 4—Diagram of LAMP system use in logic-circuit design.

In addition to the basic circuit information, it is possible to input physical design information through the LSL-LOCAL language. When the designer is satisfied with the design of the circuit on a circuit-pack basis, the verified logic is then used as a base for the physical design process. Here the various additional machine-aided tools are used to perform partitioning, placement, and routing. The successful completion of physical design thus establishes a logical and physical design data base from which other uses of LAMP in the design process may take place. Some examples of these activities are: (*i*) derivation of circuit-pack diagnostic tests for manufacturing check-out purposes;

(*ii*) design verification of the subsystems (which are formed by combining circuit packs) and the complete processor (which is formed by combining the subsystems); and (*iii*) design and verification of diagnostic program(s) and generation of TLM data.

V. EXAMPLES OF LAMP SYSTEM USE

To provide some insight into the use of the LAMP system, a few examples of simple procedures performed with the LAMP system are presented. Because of the large number of ways the LAMP system is utilized, it is impossible to cover more than a small area of the system functions. The examples shown, however, are representative of typical activity.

All user communication with the LAMP system is by use of a command language. Each command represents an action to be taken by the system. In conversational use, the system prompts for the next input by means of a > character. Some commands which require additional information prompt the user with an @ character.

Example 1—Logic Verification Run
(TSS Log-on Procedure)

```
System:  LAMP DESIGN AUTOMATION SYSTEM
         ENTER COMMANDS
         >
   User:  load expl. tables
System:  CKTNAME: EXAMPLE.CIRCUIT VERSION 06/24/73
         >
   User:  run tval expl.test.vector,expl.output.results,p
System:  LAMP TVAL SIMULATOR—VERSION 2.5
         >
   User:  display values,t
System:  AT INPUT NO. = 3
         INPUTS:   SA     SB     CA     CB
                   SEN    CEN
         OUTPUTS:  SOUT   COUT
         INPUTS:   100001
         OUTPUTS:  11
         >
   User:  end
```

In this example, the user desires to test the "good" operation of his logic design by exciting his circuit with a series of prestored input vectors. The circuit description has been previously compiled from an

LSL-LOCAL encoding into a data set called "expl.tables." The prestored input vectors are located in the data set "expl.test.vectors." Since he is not interested in fault analysis, the TVAL (true-value) simulator is chosen. For nonfaulted operation, this simulator is the most efficient of the six available. The results he needs for his analysis can be obtained in two ways. The bulk of the output is produced via the computer high-speed printer. The particular types of results the user wants are specified by the contents of data set "expl.output.results." The "p" indicates that the results are to be printed as soon as possible. Because the user wants a quick check of some of the results before the other output is available, he instructs the system to display the input and output gate names along with their associated output values on the terminal after the simulation is completed. Satisfied with the results, he ends the simulation.

Example 2—Creation of the Controlling Data Sets
(TSS Log-on Procedure)

```
System: LAMP DESIGN AUTOMATION SYSTEM
        ENTER COMMANDS
        >
  User: source lsllocal expl.source,expl.tables
System: LOCAL LP START
          LOCAL LANGUAGE PROCESSOR—VERSION 3
        LOCAL LP END
        >
  User: results expl.output.results
System: ENTER SIMULATION RESULTS SPECIFICATIONS
        @
  User: after input *every; values
System: @
  User: [default]
System: >
  User: invec expl.test.vectors
System: ENTER INPUT VECTORS
        @
  User: t'101031'
System: @
  User: t'100001'
System: @
  User: [default]
System: >
  User: end
```

In this example, the user creates the data sets used to control the simulation run shown in Example 1. The first action is to compile the logic-circuit description written in LSL-LOCAL that has been stored in data set "expl.source" in a form that the compiler can use. The compiled information is stored in data set "expl.tables." Next the data set ("expl.output.results") that controls the output results is formed by use of the RESULTS command. The information put into this data set will instruct the simulator to print the *values* of the inputs and outputs after every input vector has excited the circuit.

Finally, the series of input vectors used to excite the circuit is created by use of the INVEC command. In this case, a series of these input vectors has been created. The input value "3" signifies a "don't care" value.

Only a few of the available commands and options have been shown. However, these should provide an idea of the ways in which the system can be used. Additional examples will be presented in the other papers of this series to illustrate specific points under discussion.

VI. CONCLUSION

Present and future designs of digital systems require computer aids during all phases of development, from initial architecture specifications to diagnostic-test design. The efficiency of these tools in performing their intended functions is of great importance, from both internal (efficiency of algorithms) and external (user convenience and usefulness) considerations. Viewed in this light, the LAMP system has been an outstanding success. The use of LAMP has been found to be cost effective in that LAMP provides the designers a convenient facility to assure design quality, to expedite error correction, and to reduce design-rework cost. LAMP also offers the designer a versatile tool to evaluate and verify the system diagnostics before hardware is committed. It has become an integral part of the design of new electronic switching systems and has strongly influenced the methodology of their design.

The other papers in the series will give more detailed descriptions of the use and design of selected portions of the LAMP system.

VII. ACKNOWLEDGMENTS

Many of our colleagues have contributed to the development of the LAMP system. Contributions made by R. E. Strebendt, R. E. Michael, and E. A. Rinaldy in the development of LSL-LOCAL, A. B. Marsh in the design of the command system, R. A. Elliott and R. B. Schmidt

in the implementation of output system, and J. R. Burnside, G. A. Raack, R. R. Riser, and F. J. Webb in the development of the OS version of LAMP are gratefully acknowledged. The authors would also like to thank J. A. Harr, W. Ulrich, and R. W. Ketchledge, and the many users for their continuous support and encouragement throughout the development of the system.

REFERENCES

1. S. Seshu and D. N. Freeman, "The Diagnosis of Asynchronous Sequential Switching Systems," IRE Trans. on Elec. Computers, *EC-11* (August 1962), pp. 459–465.
2. S. A. Szygenda, "TGAS2—Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic," Proc. ACM-IEEE Design Automation Workshop (June 1972), pp. 116–127.
3. B. H. Scheff and S. P. Young, "Gate Level Logic Simulation," in Design Automation of Digital Systems, Vol. 1, edited by M. A. Breuer, New Jersey: Prentice-Hall, 1972, pp. 101–172.
4. S. G. Chappell, "LAMP: Automatic Test Generation for Asynchronous Digital Circuits," B.S.T.J., this issue, pp. 1477–1503.
5. H. Y. Chang and G. W. Heimbigner, "LAMP: Controllability, Observability, and Maintenance Engineering Technique (COMET)," B.S.T.J., this issue, pp. 1505–1534.
6. S. G. Chappell, C. H. Elmendorf, and L. D. Schmidt, "LAMP: Logic-Circuit Simulators," B.S.T.J., this issue, pp. 1451–1476.
7. T. T. Butler, T. G. Hallin, J. J. Kulzer, and K. W. Johnson, "LAMP: Application to Switching-System Development," B.S.T.J., this issue, pp. 1535–1555.
8. S. Seshu, "The Logic Organizer and Diagnosis Programs," *Report R-226*, Coordinated Science Laboratory, University of Illinois at Urbana (AD-05927).
9. D. K. Chia and M. Y. Hsiao, "Boolean Difference for Fault Detection in Asynchronous Sequential Machines," IEEE Trans. on Computers, *C-20* (November 1971), pp. 1356–1361.
10. G. R. Putzolo and J. P. Roth, "A Heuristic Algorithm for the Testing of Asynchronous Circuits," IEEE Trans. on Computers, *C-20* (June 1971), pp. 639–647.
11. R. W. Downing, J. S. Nowak, and L. S. Tuomenoksa, "No. 1 ESS Maintenance Plan," B.S.T.J., *43* (September 1964), pp. 1961–2020.