LOGCAP is a widely used system of programs that support digital logic design and testing. It has been designed and implemented twice in a period of five years. Since the last redesign three years ago, thirteen releases have kept LOGCAP abreast of the technology it serves.

In addition, a new multifault deductive simulator has been added to perform the grading of test programs, fault discrimination, fault dependent instability analysis and other fault related functions. This paper is the first public discussion of the LOGCAP system. We try to present several viewpoints in parallel within the paper. Logic design, test program generation, the design of LOGCAP, user support and program maintenance are all represented. LOGCAP serves both the LSI and PC board oriented designers.

The LOGCAP logic system comprises a large number of programs and associates with each network a large number of files. It is not the structure of files and programs how they are used with an explanation of LOGCAP. Rather we need to look at LOGCAP first from the viewpoint of a logic designer as a network progresses from conception through simulation and prototype to manufacturing and testing. Distinct features of LOGCAP come into play at each stage of this process. We can therefore use the natural evolution of the project to explain what LOGCAP is and what it does. It is only from this angle that the interrelations of certain features of LOGCAP can be appreciated. After this discussion we take up the software engineering, relative cost and future of LOGCAP.

NETWORK DESCRIPTION

LOGCAP provides a list of twenty-six gate types, each of its own intrinsic model, from which any digital network may be constructed. Each of the models corresponds to a unique block of code which determines the response of that model to its stimuli for all states of the model. The models include the combinatorial gates (NAND, AND, OR, NOR, XOR, AOI, OAI), several variations of flip-flop (DFF, SRF, JKF, JKFFD, JKFFAC), MOS and bus oriented models (MOS Transfer Gate, Tri-state Gates, Tri-state Bus), memory devices (RAM, ROM, and Shift Register) and special purpose devices such as the One-Shot and general purpose Counter. The internal models in LOGCAP mimic the device functions. Combinatorial device models are low level models composed of and's, or's, etc. More complex device models are not combinations of simpler models. Rather, they are higher level descriptions of addressing, counting, etc. These higher level models run faster than combinations of simpler devices and account for the rough proportionality of computing cost to node count in LOGCAP. Of course, all of the more complex devices can be modeled by the designer in terms of simpler devices. The relative speed advantage of the higher level models is not the only reason for using them. Network modeling occurs early in the development project. To verify concepts, the designer may model the network at the highest possible level. The designer may be to ensure that particular realization meets the requirements of a conceptual model. For this, a portion (SUBNET) of the original network may be modeled in great detail. Once the various SUBNET's have been checked, the designer may return to high level modeling. The PC Board oriented designer may never wish to model the internal details of many devices. Modeling flexibility is one of the tools that can speed a project to completion.

A brief reference was made above to SUBNET's as part of networks. The designer using LOGCAP may define and give unique names to any number of SUBNET's in a manner similar to networks. Within any network or SUBNET, any other SUBNET may be inserted by name. Recursion is not permitted. One SUBNET may include another to a nested depth of seven. LOGCAP performs decomposition and insertion of SUBNET's which increases the size (number of nodes) of the network and reduces the execution time.

The LOGCAP logic system comprises a large number of programs and associates with each network a large number of files. It is not the structure of files and programs how they are used with an explanation of LOGCAP. Rather we need to look at LOGCAP first from the viewpoint of a logic designer as a network progresses from conception through simulation and prototype to manufacturing and testing. Distinct features of LOGCAP come into play at each stage of this process. We can therefore use the natural evolution of the project to explain what LOGCAP is and what it does. It is only from this angle that the interrelations of certain features of LOGCAP can be appreciated. After this discussion we take up the software engineering, relative cost and future of LOGCAP.

NETWORK DESCRIPTION

LOGCAP provides a list of twenty-six gate types, each of its own intrinsic model, from which any digital network may be constructed. Each of the models corresponds to a unique block of code which determines the response of that model to its stimuli for all states of the model. The models include the combinatorial gates (NAND, AND, OR, NOR, XOR, AOI, OAI), several variations of flip-flop (DFF, SRF, JKF, JKFFD, JKFFAC), MOS and bus oriented models (MOS Transfer Gate, Tri-state Gates, Tri-state Bus), memory devices (RAM, ROM, and Shift Register) and special purpose devices such as the One-Shot and general purpose Counter. The internal models in LOGCAP mimic the device functions. Combinatorial device models are low level models composed of and's, or's, etc. More complex device models are not combinations of simpler models. Rather, they are higher level descriptions of addressing, counting, etc. These higher level models run faster than combinations of simpler devices and account for the rough proportionality of computing cost to node count in LOGCAP. Of course, all of the more complex devices can be modeled by the designer in terms of simpler devices. The relative speed advantage of the higher level models is not the only reason for using them. Network modeling occurs early in the development project. To verify concepts, the designer may model the network at the highest possible level. The designer may be to ensure that particular realization meets the requirements of a conceptual model. For this purpose, a portion (SUBNET) of the original network may be modeled in great detail. Once the various SUBNET's have been checked, the designer may return to high level modeling. The PC Board oriented designer may never wish to model the internal details of many devices. Modeling flexibility is one of the tools that can speed a project to completion.

Each device which the designer chooses to represent corresponds to one instance in which one of the LOGCAP models is used. Each model has one or more inputs and one or more outputs. A key number in measuring the size of a network is the number of nodes in the network. In LOGCAP only the outputs of the models count as nodes, one for combinatorial gates, two for flip-flops, six for the counter, one for the shift register, and as many as the designer specifies for the RAM and ROM. The simulation capacity of LOGCAP is limited to one hundred nodes. No ready comparison can be made with other programs that depend for part of their computations on NAND-Gate representations. For example, the number of internal NAND Gates required to model a RAM with a certain number of outputs depends upon the number of address lines. Thus, the second capability which LOGCAP offers designer is sheer capacity, exceeding by almost one order of magnitude the demands of present day networks. Most LSI designs do not exceed a few thousand nodes and a PC Board with 100 devices may be only a few hundred.

Timing considerations are as pervasive in modeling as in practice. For each instance of use, each model must be assigned a relative delay. LOGCAP follows the designer practice of providing a time interval of no intrinsic duration which the designer scales to meet his needs. Separate rise and fall delays, expressed in intervals, must be assigned to all devices. The fall delays are fixed but may be made fan out dependent for those networks which require such consideration. Two distinct approaches to delay modeling are common. Delays may be modeled in detail by scaling the interval to the greatest common factor of all the delays present in the network. Each device rise and fall delay is expressed as a multiple of the interval unit. Accurate timing analysis is obtained with this method. If only logical results are wanted, however, most of the delays in a network can be ignored in LOGCAP. This is done by setting rise and fall delays to zero for most devices. The common name for such a model is a minimum delay model. Minimum delay modeling is not only cost effective, it has particular advantages in the analysis of the network and in detecting races and hazards. We return to these topics later and take up the network analysis implications here. A feedback loop is a common occurrence in logic networks. For large networks the designer may not notice an error which produces unwanted feedback. A loop may oscillate in practice and so may a simulated loop. LOGCAP performs in each interval all of the computations needed by all devices which are excited but having no delays pending. If no device in a loop has delay and one of them is excited, a contradiction occurs. LOGCAP either cannot begin computing a device state because all of its inputs are not known, or cannot stop computing devices around the loop because the effect of previous computations never scope propagating. To prevent this situation, LOGCAP requires that at least one device in each loop have a positive delay. LOGCAP performs a topological analysis to locate and flag any loop in which the designer did not assign a delay to at least one of the models. In minimum delay modeling, the designer makes zero all of the delays except for a device in each of his intended feed back loops. LOGCAP can then tell whether any signals are propagated before any simulation is performed. It is a simple matter for the designer to add delays wherever needed for subsequent modeling.

A brief reference was made above to SUBNET's as part of networks. The designer using LOGCAP may define and give unique names to any number of SUBNET's in a manner similar to networks. Within any network or SUBNET, any other SUBNET may be inserted by name. Recursion is not permitted. One SUBNET may include another to a nested depth of seven. LOGCAP performs decomposition and insertion of SUBNET's which increases the size (number of nodes) of the network and reduces the execution time.
There are several applications of SUBNET's in conjunction with other LOGCAP features. First, of course, is the saving of a designer's time for modeling repetitive logic. Second is the ability to model and test a network and save it as a tested SUBNET for use in other networks. Third is the built in capacity of LOGCAP for creating a library of private SUBNET's for each designer, group or company. The modularity of multi-device PC Board networks and the standard cell approach to LSI design are served by this library feature. LOGCAP provides two commercial device libraries containing pretested models. The designer need not separately store these models. LOGCAP will search the libraries as the models are needed. One of the libraries contains commonly used TTL and MOS devices. The models in this library can be printed, resaved and altered as private models by any designer. The second library provided by LOGCAP is a proprietary library. It contains models of proprietary devices made by vendors such as AMD for the use of LOGCAP designers. Models in this library cannot be printed by designers, but are inserted by LOGCAP into any network as called for by the designer. The purpose of the proprietary library is to solve two classes of problem. First, the custom LSI vendor-customer interface in which the standard cells may be proprietary but the custom needs to use them in simulation. The other problem is the difficulty designers face in designing around LSI devices like the MC2901 and micro-processors. Here again a partial conflict between the proprietary property interest and the logic designer's need to know can be resolved. Since the LOGCAP staff works with the vendor to assure proper modeling, the logic designer can have confidence in the fidelity of the model. The overall impact of the public and proprietary libraries is only now becoming apparent. We are now seeing companies use LOGCAP to specify parts to LSI vendors with good acceptance by the vendors. The potential for improved communication and project economy is clear.

There is one more feature of LOGCAP network modeling whose significance will become clear later in the context of simulation and post-simulation analyses. Each node which the designer defines is given a unique name. The designer refers to nodes by name when requesting to printed data, etc. Internal nodes within SUBNET's receive alias names created by LOGCAP. A SUBNET option allows the designer to suppress creation of alias names for the internal nodes, leaving them unnamed. Later, when LOGCAP performs various analysis, nodes with alias names are ignored. In such cases, the designer not only saves simulation expense, but also the power of LOGCAP can be concentrated on doubtful portions of the network. This topic is explored in more detail where the various analyses are discussed.

In summary, it is not enough merely to be able to describe a logic network. Today's networks are becoming extremely complex. The designer needs flexibility, pretested models, and a convenient means of directing the analysis within the network, all of which LOGCAP provides.

PATTERN DEFINITION

The terminology of network description is fairly common, but the terminology for describing network stimuli is not. To begin with, LOGCAP introduced a second unit of time, the CYCLE, to define the time between changes of input stimuli. The designer specifies the number of intervals in a CYCLE. This usage conforms to the natural description of networks like micro-processors which have an internal cycle of some number of intervals. In the context of test program generation, we would say that inputs change at the beginning of a CYCLE and outputs are strobed at the end of the CYCLE. The sequences of states through which the designer wants to drive the input nodes are called patterns. They are external data supplied by the designer. An examination of common input patterns for digital logic networks reveals three general cases. The first are random patterns in which an input is high or low most of the time an only occasionally in the other state. For this case, LOGCAP provides a simple exception description. The designer specifies the input node name, the exceptional state (HI or LO) and the CYCLE numbers for which the input takes that state. It is understood that for all other (unmentioned) CYCLE's the input is in the opposite state.

The second case of input patterns is arbitrarily phased clock patterns on one or more inputs. LOGCAP provides a simple repeat syntax in which the designer briefly describes these patterns. The third case of input patterns is a group of inputs, treated as a unit to which is applied an overall pattern. Examples of such patterns are random, walking one, walking zero, gray code, count up and count down. For each case, LOGCAP provides a brief syntax in which the designer may describe how a situation may be described. The purpose of these variations of pattern syntax is to ease the task of description, reduce input errors and make the patterns instantly readable to the designer.

It frequently happens in practice that a simulation is carried to a certain point (CYCLE number) and further progress depends upon the designer's review of prior results. The designer often changes the input pattern description several times after the network description is final. This practical necessity is reflected in the LOGCAP stop-start-continue design. At any interruption point, LOGCAP creates a restart point. The designer may later return to any restart point and resume computation without repeating earlier work. This basic principle permits the accumulation of correct results. As networks become more complex and it becomes impractical to predict the whole course of simulation, economical segmentation becomes necessary. LOGCAP provides all the file maintenance required automatically and transparently.

The last feature of the pattern description to be discussed is initialization. LOGCAP does not initialize a network to well defined states automatically. It starts with all nodes in the undefined (X) state and presents to the designer the initialization problem. The designer may not want to simulate his solution to the initialization problem for networks with predictable states and log initialization sequences. LOGCAP provides for this case with the ability to set or reset any nodes in the network at any restart point including initially. This technique is also used to avoid the simulation of barren stretches in digital watches and other networks with long counter chains.

SIMULATION

LOGCAP simulation is controlled by a language whose commands may come either from the designer at a terminal or from a control file. The simulation process is divided into three major parts for logic simulation and three corresponding parts for fault simulation. Each function in the system responds to a distinct network context. The designer uses the command language to tailor each run to the need of the moment. The complexity of learning additional language is the cost of fit to function. LOGCAP commands can typically be learned in a day and mastered in a week. For logic simulation, the fit is achieved by compiling the network description with its models, delays, SUBNET's, and library references into a data structure that speeds up simulation. The second step is simulation proper. Here data is printed on the fly, a history
of the network is stored, and the stability analysis of the network is performed. As pointed out earlier, the standard logic analysis in LOGCAP required that the network be stable before the inputs are allowed to change. A designer may sometimes want to change the input pattern before the effects of the previous change have stabilized. In this case the designer may override the stability requirement and force inputs to change synchronously. Special techniques are available in LOGCAP to diagnose a network which fails to stabilize properly. During simulation the four possible states (0, 1, X, and Z) of each node are computed and stored in a HISTORY file either cycle by cycle or interval by interval as the designer specifies.

**ANALYSES**

After simulation several types of analysis may be performed using the saved data without repeating the simulation. The designer may extract the traces of any nodes for any cycles or intervals. This is frequently the best way to trace the effects of certain signals through the network. Printed FANIN and FANOUT lists are always available to assist screening. Usually, the designer will specify input patterns to exercise the logic functionally according to its design. Selective print out is the first tool of design verification. If the network operations have been fully exercised, all of the nodes should have toggled, i.e. taken both a one and a zero value. In practice we find that when all of the nodes have toggled and the network has been functionally excited, fault verification will show a coverage of 85% or better. For this reason LOGCAP provides the economical VERIFY TOGGLES feature. There is little point in grading a test program for fault coverage until all the nodes toggle. Experience shows that detection is too low.

Timing analysis can be useful quite early in the simulation. There is a competition in designing test programs for high volume production between short test programs and unambiguous test programs. Programs can be shortened by changing many inputs simultaneously at the risk of introducing races and hazards. If only one input is changed per cycle, then any races or hazards that occur are clearly generated within the network. On the other hand, the test program may become impractically long. Our experience is that test programs for many networks can detect more than 90% of the stuck faults with races and hazards in cycles which are less than the number of nodes. However, dense test programs must be checked for races and hazards. The VERIFY RACES and VERIFY HAZARDS commands perform these functions in LOGCAP. As mentioned earlier, delay modeling interacts closely with race and hazard detection. In some programs, delays are modeled very closely and varied during simulation to detect races and hazards. While detailed delay modeling is often useful, there is a more cost effective method for most cases. The designer simply uses minimum delay modeling and the VERIFY RACES and VERIFY HAZARDS commands. In a minimum delay network model, all of the delays are made zero except as necessary in feedback loops. This forces near races and hazards to become exact races and hazards which are easily detected. Further, the VERIFY RACES and VERIFY HAZARDS commands can be applied either on a cycle by cycle basis or on an interval by interval basis according to the stored network history. The resulting analyses will call attention to potential races and hazards which the designer should examine in more detail. Upon further examination using selective printing, the designer can determine whether any of the candidates discovered by LOGCAP is a real problem.

Another type of post simulation analysis is single fault simulation to verify detection. Single fault simulation is so named because the simulation is run for each fault until some network output produces a result which contradicts its fault free state. Single fault simulation is used to evaluate changes in the test patterns intended to detect certain faults. There are two occasions for such analysis. First, early in the development of a test program, the designer may want to sensitize a path before exercising a block of logic that drives it. One fault can be chosen such that when it is detected, the path is known to be sensitive. Then the remaining faults can be exercised out. The second case occurs when a set of functional patterns has been graded and the designer has a list of undetected faults. Patterns must then be added to the end of the pattern set to increase the percent detected.

All of the analyses described above apply only to nodes that have names. As mentioned earlier, the designer may optionally suppress names within SUBNET's. Since the designer may also use SUBNET's freely in the network description, it follows that all of the analyses above may be aimed at selected parts of the network. This feature comes into play when a designer is using well tested SUBNET's to construct a part of a larger IC or PC Board. Modular design, standard cells and iterative logic are becoming more common. By taking advantage of these techniques a designer can reduce his total design effort. The designer who suppresses the node names in well understood portions of the network not only saves the cost of those analyses. He can also focus attention on parts of the network without the distraction of large quantities of extraneous data. In design work, unwanted data is not valuable. It is noise. LOGCAP provides noise suppression. Of course, nodes without names are entirely valid in all other respects. Signals and faults propagate through them. They are merely inaccessible for certain functions.

**MULTI-FAULT SIMULATION**

Once the correct logical functioning of a network has been verified using the tools described above, design attention turns to the testability of the network. LOGCAP multi-fault simulation can be used to investigate the detectability of faults, their output signatures and some of their effects upon network behavior. The multi-fault programs use as input data the information provided by logic simulation, a compiled network description and input pattern description. LOGCAP analyzes six fault classes, stuck-at-one and stuck-at-zero for inputs and outputs of all device models and complementary stuck faults for flip-flops. In the latter cases one output of a flip-flop is stuck high while at the same time the other output is stuck low. The first step in multi-fault simulation is to DEFINE the faults that may be simulated. The DEFINE command invokes a program which defines and classifies all faults in the six classes for the network. Nodes without names do not contribute faults. For all named nodes, faults are classified as undetectable, equivalent or real. Real faults are those that must be simulated to obtain correct fault coverage data. An equivalent fault will be detected if and only if some real fault is detected. LOGCAP identifies the pairings of equivalent and real faults as a result of fault collapsing. Undetectable faults are topologically concealed and are not simulated. Results of the fault definition analysis are saved for selective examination and for other purposes during fault simulation. About 10% to 40% of faults are collapsed out for highly sequential and highly combinational networks respectively.

The volume of data that can be produced by a logic simulation is multiplied by the number of faults in a faults simulation. In the fault simulation phase of a project, the designer typically needs statistical summary data on the fault population and exception data on certain groups of faults. Large amounts of printed
SOFTWARE ENGINEERING

The algorithms, techniques and code generation methods employed in LOGCAP provide an example of modern software engineering applied to the initial design, maintenance and enhancement of a major software system. The system is optimized for virtual memory and direct access disk I/O operations. It executes interactively or as a batch program at the user's option. A good mix of interpretive and compiler techniques is used to achieve flexibility and speed of execution. For example, the commands are interpreted and those tasks that require relatively little computation are performed immediately by the appropriate routines. Network reordering, compilation, SUBNET insertion and alias generation are performed by a set of programs that create data structures optimized for execution speed in logic simulation. The simulation itself is performed over these data structures by programs designed for speed rather than size. Certain functions of the system would cost more to execute than the value of the data produced if conventional programming methods were employed to generate the programs. In these cases LOGCAP compiles absolute machine code to match the particular problem at hand. This machine code is then executed to achieve the ultimate in execution speed. Elsewhere, extensive use is made of compiler methods, symbol tables, list processing, etc.

Models for all of the intrinsic devices in LOGCAP are based directly on the logic designer's understanding of how they work. Models for complex devices are not built up out of lower level models. Look ahead is used to assure that devices and intervals are skipped when no signals are changing. Similar methods are applied in the fault simulator design. The fault simulator is in the class which has been called deductive in the literature. We feel that a more accurate description might be to call it a computer of first logical differences. All paths are treated as feedback paths thereby avoiding special analysis for the feedback case. Multiple paths in which the same fault may arrive at several device inputs are handled by computing the device state for an interval if one or more of its inputs has changed. Multiple evaluations may occur within a cycle. For devices without delay, only the correct data remains at the end of a cycle. For devices with delay, a complete delayed representation is maintained and updated as required by signal changes at the inputs. The principal design problem in the fault simulator was the complexity of algorithms for fault dynamics and delayed first difference processing in memory devices. These methods were designed and checked to be logically equivalent to the simultaneous simulation of many networks each with a single fault. Fault models are used for each of the intrinsic logic models in LOGCAP including the RAM, ROM and counter.

It is sometimes argued that programs must be written in assembly language to gain speed. LOGCAP is a good counter example. There are about 1000 lines of assembler code and about 20,000 lines of Fortran in the system. The assembler code is used in small routines to perform subfunctions not directly accessible from Fortran. The principal algorithms are not coded in assembler either in the logic or in the fault simulator. Neither are they coded in Fortran directly. The significant algorithms of LOGCAP are written in a structured language called DESTRAN for DESign TRANslation. The object code produced by one of the DESTRAN compilers is Fortran. The DESTRAN language symbolizes whole programs, their structure and flow of control. DESTRAN differs from analytic languages like FORTRAN in two important ways. First, the natural evolution of a program in DESTRAN code is from the general and abstract to the concrete and particular. This orientation fits product design, project planning, software maintenance
and documentation better than opposite orientation derived from analytic languages. The second is that in DESTRAN there are two source files. One defines the structure of the program. The other controls the details of execution code and flow of control at the lowest level emitted by the compiler. It is the appropriateness with which algorithms can be described in DESTRAN which one or two people to write and maintain a system of this size over a number of years and through many releases.

The documentation and human interfaces to LOGCAP are important to its practical application. In addition to a user's guide and periodic release notices, there are application notes, an introduction and an overview available. The National CSS timesharing system provides literature and local assistance nationwide. An online system of HELP files is accessible to the terminal during a session to prompt the user regarding details of syntax, command sequence, etc. The customer and technical representatives of National CSS provide the first level of personal assistance. The LOGCAP staff provides two more levels of help for any operational or technical considerations that may arise.

**BENCHMARKS**

The logic designer who knows his network can predict LOGCAP costs with good accuracy. Each function of LOGCAP charges per unit of work done. LOGCAP works pays for the computer resources used, except for the connect time. Costs are significant only for large networks. Since large runs should be made as batch jobs and submitting a batch job is quite easy, the costs quoted will be based on batch execution. There is no connect time charge for batch runs, so LOGCAP charges are predictable. The LOGCAP charge for network compilation is $20 per thousand nodes. As an example, the AM2901 chip in the proprietary library is only 189 nodes. The number of nodes in a network is easily estimated and this count is printed by LOGCAP when the network is compiled.

Estimating simulation cost is a bit more complicated. The basic charge is 8h per 100,000 node-intervals.

**The logic designer who knows his network can predict LOGCAP costs with good accuracy.** Each function of LOGCAP charges per unit of work done. LOGCAP works pays for the computer resources used, except for the connect time. Costs are significant only for large networks. Since large runs should be made as batch jobs and submitting a batch job is quite easy, the costs quoted will be based on batch execution. There is no connect time charge for batch runs, so LOGCAP charges are predictable. The LOGCAP charge for network compilation is $20 per thousand nodes. As an example, the AM2901 chip in the proprietary library is only 189 nodes. The number of nodes in a network is easily estimated and this count is printed by LOGCAP when the network is compiled.

This charge is based on the observation that the computation cost is roughly proportional to the number of nodes for each model in LOGCAP. Since logic simulation has its purpose to generate the state of each node for each interval of simulation, the product of the number of nodes by the number of intervals simulated is a measure of the work done. Only those intervals are counted in which computation is required. The number of nodes in the network is known to the designer. The designer can estimate (or for LOGCAP compute) the number of intervals in each cycle required for the network to stabilize. The designer also determines the number of cycles to be simulated. A reasonable estimate can therefore be made for the simulation cost. Again using the example of the AM2901, a simulation of 295 cycles, enough to exercise it thoroughly, costs about $8. This corresponds to a cycle length of one or two intervals, a good example of the economy of minimum delay modeling.

Estimating simulation cost is a bit more complicated. The basic charge is 8h per 100,000 node-intervals.

This charge is based on the observation that the computation cost is roughly proportional to the number of nodes for each model in LOGCAP. Since logic simulation has its purpose to generate the state of each node for each interval of simulation, the product of the number of nodes by the number of intervals simulated is a measure of the work done. Only those intervals are counted in which computation is required. The number of nodes in the network is known to the designer. The designer can estimate (or for LOGCAP compute) the number of intervals in each cycle required for the network to stabilize. The designer also determines the number of cycles to be simulated. A reasonable estimate can therefore be made for the simulation cost. Again using the example of the AM2901, a simulation of 295 cycles, enough to exercise it thoroughly, costs about $8. This corresponds to a cycle length of one or two intervals, a good example of the economy of minimum delay modeling.

When detailed time delay modeling is used, the cost of logic simulation rises in proportion to the number of intervals of delay in the longest excited path assuming that gate activity is spread uniformly over the cycle. The network topology also affects simulation cost since multiple feedback loops with delay present in the network, then combinatorial networks without delay. The last factor in the cost of logic simulation is the nature of the input patterns. Sequences that excite activity in complex gate structures will require more intervals for the network to stabilize. To cite another example, a network of 300 nodes simulated for 2000 cycles cost $100, implying about two intervals per cycle.

The various analysis functions have their own rates. For example, the simulation cost is the major cost area in LOGCAP. Fault simulation is the other major cost area in LOGCAP. Fault definition costs $4 per thousand faults. This task is comparable in scope and cost to the network compilation. The AM2901 model has 1185 faults and the other network mentioned above has 2727. Of these faults, 841 and 1901 are real, respectively. The number of cycles required to detect a given percent of these faults depends on the particular input test sequences. Test programs for detecting faults tend to be much easier to be generated by a designer than when automatic test generation programs are involved. Shorter programs cost less to simulate and reduce the testing cost per unit particularly in manufacturing applications where most of the units are good. Consider the cost to grade a test program, i.e., to determine the percent of faults detected and identify those that are not detected. Experience shows that the grading simulation for a 90% detection test program will cost 20 to 50 times as much to simulate as a single true logic simulation for the same pattern and network. Why the variation? LOGCAP ceases simulating and charging for a fault as soon as its detection is assured. A good test program detects many faults early and so reduces the amount of simulation to be done. A poor test program forces LOGCAP to carry more faults all the way through.

The required length of a test program to detect a fixed percent of faults can often be related to the network topology and number of nodes. We discuss only test programs produced by designers. A network for which a 90% test program requires a number of cycles smaller than the number of nodes will be called relatively "open." When the number of cycles required exceeds the number of nodes, the network will be called relatively "closed!" These terms are useful only for networks of significant size. A relatively open network is characterized by shallow loop structures and a relatively large number of inputs and outputs. The short sensitized paths and potential for parallel testing of devices shorten the overall test. Most PC Boards are relatively open networks. A relatively closed network has fewer inputs and outputs compared to the number of nodes. It contains loops within loops, long counter chains, or long state transition chains of other kinds. Examples include digital watches and calculator chips.

Test programs intended for fault discrimination present a slightly different situation. The best discrimination programs are also based upon the designer's knowledge not only of the network and its functions, but also of the test equipment to be used. The test program may be somewhat longer. The simulation will cost about 40% more to save all of the fault signature information and the large volume of information created can make printing a significant expense. In all cases, the ability with LOGCAP to add a few more patterns with only incremental expense is most important.

When comparing bench marks among different logic simulation systems, a few cautions are necessary. There are no standard test networks and they would change over time as technology changes. We expect that the LOGCAP libraries will come the closest to serving this function. Since networks with differing topologies and devices in them may interact differently with different simulator designs, it may not be practical to project experience with one network onto all networks. Even for a single network, it is important to be sure that the problem is properly reformulated it terms optimum for each system. For example, LOGCAP uses two units of time. Most other systems use one unit of time. A proper comparison may require reconsideration of the description of patterns, the data saved and the information printed. As programs change in response to technological demands, some of the features evaluated may lose or gain value.
New features or computational techniques are added to LOGCAP every few months in response to users needs. In several instances, benchmarks several months apart have yielded results increasingly favorable to LOGCAP.

Another consideration in comparison benchmarks is to be sure that the programs perform the same tasks or that the differences are accounted for. The completeness and flexibility of LOGCAP make it easy to do exactly what a designer needs to the moment. More monolithic programs may lack this flexibility. When comparing NAND gate equivalent simulation, the higher level models in LOGCAP should be considered. In terms of NAND gates, $100 worth of simulation presently represents about 156,000 node-intervals in LOGCAP. However, if the network consists of JK flip-flops, the same cost will purchase the equivalent of about 600,000 NAND gate equivalent node-intervals. Shift registers, counters and other devices may have higher multiples.

Focusing on the details of benchmark analysis may blur a much larger and more economically important view, that of the project as a whole. Logic and fault simulation is now an important and cost effective methodology. The most cost effective system is the one that has what the logic design project needs. The most cost effective functions of LOGCAP are those which LOGCAP makes unnecessary.

THE FUTURE OF LOGCAP

There is always an active list of new features that LOGCAP users want in the system. This list is constantly reviewed and those features that are generally useful become the basis for a new release. Additional models in the LOGCAP intrinsic library are actively being considered. There will certainly be expansion of the LOGCAP proprietary and public libraries. The control language is to be extended to accommodate conditional statements that are data dependent. Translators for other equipments and additional programs to perform new functions are also under consideration.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of National CSS, Inc. for the computer time required to prepare this paper. Our thanks to American Micro Devices for permission to mention their products and assistance in verifying models. We are also indebted to Sue Ungermann for testing the features of LOGCAP, assisting designers who are using LOGCAP and many original contributions to its present configuration.