LEVEL II (12)

RADC-TR-78-233
Final Technical Report
November 1978

INTRODUCTION TO D4LASAR, A TEST GENERATION SYSTEM

R. W. Heckelman

General Electric Company

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-233 has been reviewed and is approved for publication.

APPROVED: Michael Lavelle

MICHAEL G. LAVELLE
Project Engineer

APPROVED: Joseph J. Naresky

JOSEPH J. NARESKY
Chief, Reliability and Compatibility Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBRM) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

This page is Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-78-233	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): INTRODUCTION TO D4LASAR, A TEST-GENERATION SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Nov 77 - Jul 78
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): R. W. Heckelman	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0167
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Electric, Electronics Park, Syracuse NY 13201	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 23380142
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (RBRM), Griffiss AFB NY 13441	12. REPORT DATE: November 1978
	13. NUMBER OF PAGES: 62
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Same

18. SUPPLEMENTARY NOTES
RADC Project Engineer: Capt Michael G. Lavelle (RBRM)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Test generation
fault simulation
D4LASAR

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This report is intended to ease the initiation of future users of the D4LASAR test-generation system at RADC/RBRM. Testing-technology background material is presented, and the operation of the major D4LASAR program modules is discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

This page is Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 943

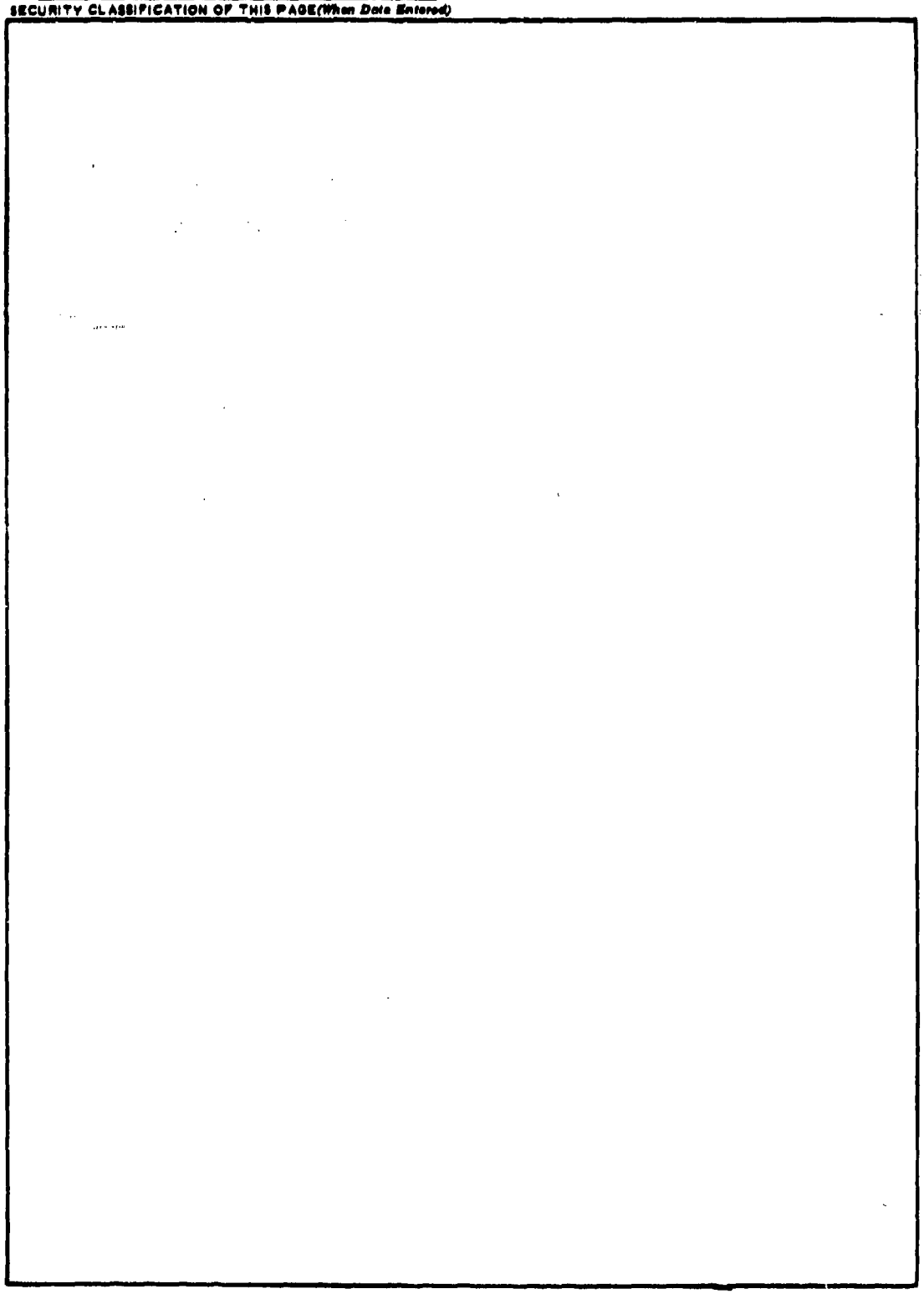
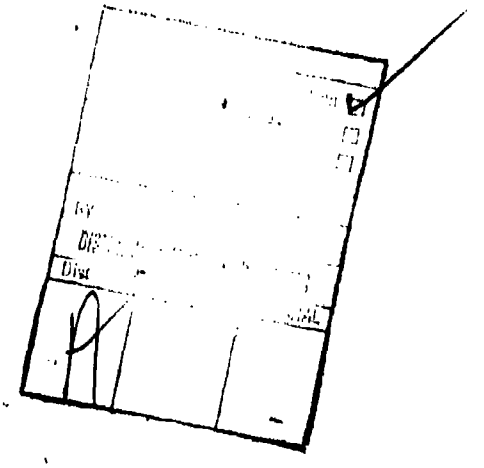


TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

LIST OF ILLUSTRATIONS



PREFACE

This report is submitted as partial fulfillment of the requirements of IITRI Statement of Work 277RAC (June 10, 1977), which is part of a larger program whose ultimate goal is to provide Rome Air Force Development Center (RADC), RBRM Section, with state-of-the-art computer aids for identifying the locations of faults in large-scale integrated (LSI) chips, and to assist in the writing of MIL-M-38510 slash sheets. The techniques were to be adapted for the RADC Tektronix S3260 Tester.

The most significant tasks of this 277RAC project were 1) training of RADC/RBRM personnel in the use of Digitest Corporation's DLASAR, Version 4, (D4LASAR) an automated test generation system and 2) development of an ISO package and installation on the RADC S3260 tester. ISO will accept tape files from the D4LASAR system and, using test data from the S3260, provide fault isolation analysis for the chip under test.

The training course on D4LASAR was primarily a series of lectures based on the Digitest Corp. users' manual combined with hand's-on experience using the G E. D4LASAR system in Syracuse, N.Y. It was preceded by discussion of the technical background underlying D4LASAR and testing in general. This report essentially covers the technical background material with the intent of easing the initiation of future users of D4LASAR.

EVALUATION

The objective of this effort was to investigate and apply structural and functional test generation techniques to LSI circuits. Both fault detection and isolation testing are of interest, the former in support of RADC TPO R5B and MIL-M-38510, General Specification for Microcircuits, and the latter in support of RADC's microcircuit failure analysis work. Both areas stand to benefit greatly from the application of computer-aided test generation as the level of integration continues to increase in digital microcircuits.

This report provides an introduction to the D4LASAR structural test generation system. Although D4LASAR was originally designed for printed circuit board testing, it is being successfully applied to LSI microcircuits. The material contained in this report will result in more intelligent application of D4LASAR to LSI testing and also serves to illustrate many of the general problems and processes involved in any automatic LSI test generation systems.

Michael Lavelle

MICHAEL G. LAVELLE, CAPT, USAF
Solid State Applications Section
Reliability Branch

I. INTRODUCTION

Most of the material presented here is intended as an introduction to D4LASAR theory. D4LASAR is a proprietary system whose detailed operation has not been disclosed by Digitest Corp, thus, some descriptions here refer to the way a job might be done rather than the manner in which D4LASAR actually does it. To reduce confusion iteration is employed. First, the D4LASAR structure and capability are outlined. Then background requirements for testing digital devices are discussed. This is followed by a further discussion of D4LASAR's application to testing, including some options. Then the operation of major analytic program modules is described and their limitations are discussed. Finally, the strengths and weaknesses of D4LASAR are discussed along with desirable evolutions and alternative approaches to test problems.

II. GENERAL DESCRIPTION OF D4LASAR

D4LASAR stands for Digitest Version 4 Logic Automated Stimulus And Response. It is a software system that executes on a dedicated SMC-3100 midi-computer for hands-on batch processing. There is also an adaptation that executes on the Univac 1100 computers of University Computing Corp. for time-shared service.

D4LASAR accepts a card-deck description of a digital device, generates a set of test vectors for it, evaluates the fault coverage, and provides a fault-dictionary printout. Manually inputted test sets may also be used.

There are seven basic subsystems plus an overriding executive subsystem and a manual-entry subsystem. The names of these subsystems should be memorized by the reader since they are referenced frequently. Figure 1 shows a typical activity flow.

Major Subsystems

INPUT This subsystem converts user description of the device to be tested into the files required by D4LASAR.

STIMGN This subsystem selects stimuli (test-input vector sequences) which will detect faults. These stimuli are usually not completely specified, i.e., they will contain don't-care values.

OVRLAY This subsystem combines test sets from STIMGN by overlaying don't-care values with specified values.

SIMUL This is a logic simulator that computes the fault-free output values for each test input vector. It also provides an analysis of races and hazards. Further, it attempts to eliminate these timing problems by inserting new "buffer" patterns between test vectors so that fewer input bits change at any one time. Failing to achieve total "deracing", it marks each affected output as a don't know.

DYSOGN This subsystem accepts the SIMUL files and provides a fault simulation. It also provides a very limited race-after-fault analysis.

REDUCE This subsystem accepts DYSOGN files, discards unnecessary test responses, and generates a three-part fault dictionary.

ISO This subsystem accepts REDUCE files and test-station failed-device data and provides fault-isolation information.

ALEC This acronym stands for Automatic Laser Executive Control. This executive module simplifies the use of D4LASAR by automatically providing many of the required job controls. ALEC also forces STIMGN, OVRLAY, SIMUL, and DYSOGN to execute in iterative cycles which considerably improve the efficiency of the process.

TECO This subsystem provides direct user control of test-input vectors, macro generation of inputs, and editing of input vector streams. It usually drives SIMUL, and it can be preceded or followed by a STIMGN-OVRLAY-SIMUL-DYSOGN cycle.

Typical Usage of D4LASAR

Figure 1 diagrams the typical control flow through the major subsystems of D4LASAR. The system selects test exercises for digital devices but does no actual testing; a separate tester is required for that. The network to be tested must be modeled as an interconnected assembly of hardware modules, and the modules must consist of gates or assemblies of gates. The Component Library contains 14 basic functions such as NAND, AND, wired AND, two-NAND LATCH, JK, RS, and D flip-flops, and MOS transmission gates. The Library also contains models for more than 500 TTL devices plus some devices of other IC technologies. However the user may specify a network model, INPUT will translate it into an equivalent NAND network for subsequent operations. Special control cards are required to designate which network nodes are accessible inputs and which are observable outputs. Two-way tri-state terminals require special modeling, which is discussed later.

STIMGN is a "back tracing" program that assigns a value to one selected output, then proceeds "backwards", i.e., opposite to signal flow, while assigning input values to gates to achieve local test objectives. A test will be a sequence of external-input values which cause a logically consistent set of internal gate-input values such that the existence of a specific failure mode will be detectable at the selected output. Usually many of the external inputs have don't-care values. Cross-coupled NAND latches are identified automatically and the sequential nature of each latch is recognized. However, STIMGN has no ability to account for asynchronous circuit delays. Combinations of latch values which STIMGN seeks to obtain, but which give rise to logical inconsistencies, are identified as "illegals". Latches composed of more than two NANDS are usually also identified as illegals. Encounter with an illegal causes STIMGN to discontinue its current search and to go to the next test objective. STIMGN stops when a time limit has been exceeded or when it attains its current goal in terms of percent detected faults. STIMGN has an unreal view of the network so it may have actually accomplished more or less than it thinks it has achieved. DYSOGN will be the final arbiter of fault coverage.

Each STIMGN test is independent of the rest in that each backtrace begins with a unique local test objective, and STIMGN develops, where it can, the initial test conditions for that test objective. It is not possible for the user to specify either an initial memory state or an initial input state. The user can specify

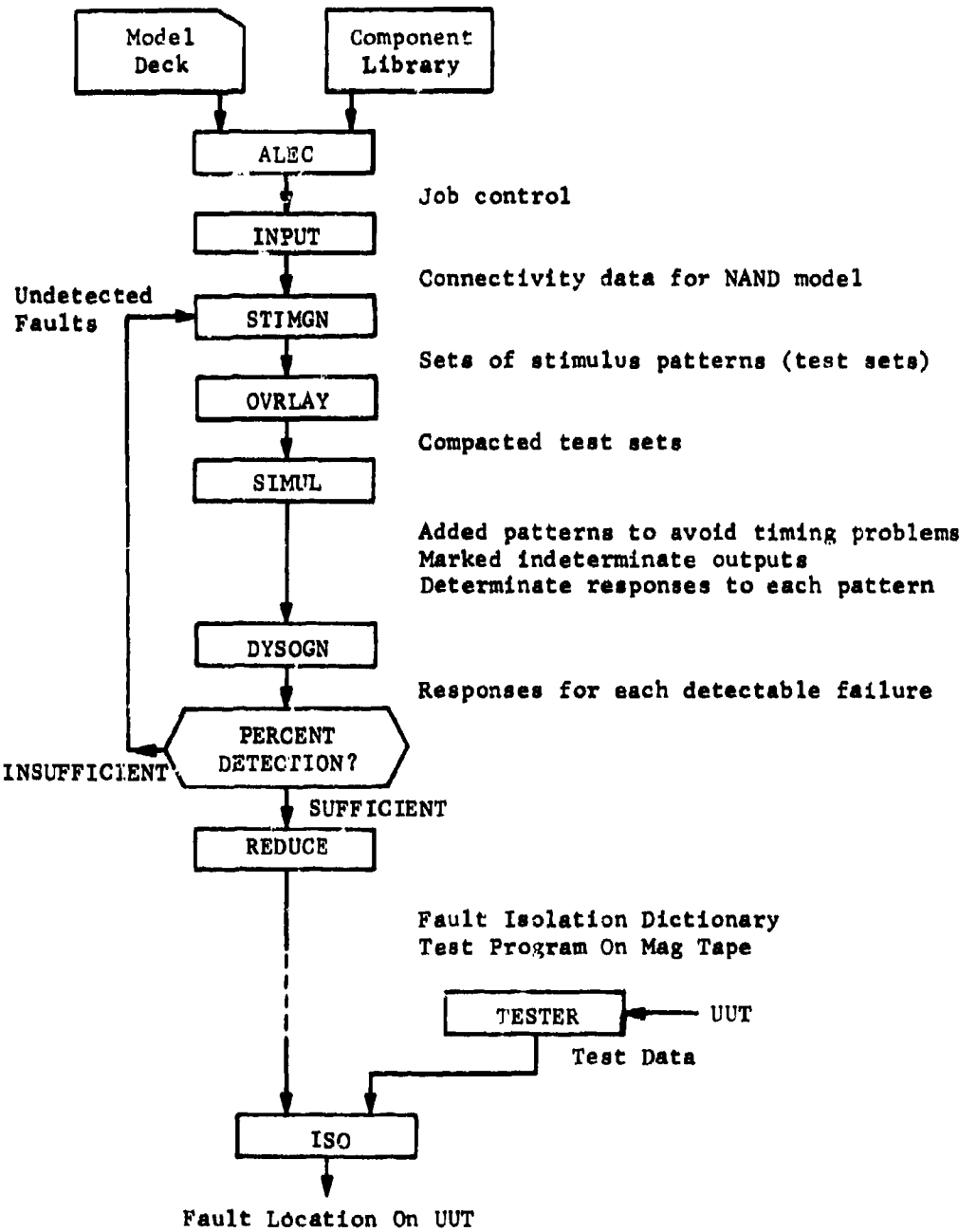


Figure 1. Typical D4LASAR Flow

"illegal" combinations of network nodes, but these forbidden combinations remain invariant throughout the D4LASAR execution.

OVRLAY takes the most recent set of tests spawned by STIMGN and merges sequences of vectors where there is no zero-one conflict in any pair of corresponding bits. This reduces the number of test sets by exercising more signal paths in parallel than STIMGN could account for. In the process, many don't-care input bits are assigned preceding or following STIMGN values. All remaining don't care bits are assigned in a way that tends to preclude races and hazards. The final output of OVRLAY is a list of binary vectors that still constitutes a sequence of independent test sets as ordered by STIMGN, but for which there remains no distinction as to where intermediate STIMGN tests begin or end.

SIMUL applies the vector sequence from OVRLAY to the NAND model of the network. This model is essentially that of the user, and it is also used by DYSOGN. For sequential networks it may be necessary to provide STIMGN with a somewhat different model to work around STIMGN's inability to account for gate delays.

SIMUL's primary task is to compute the fault-free responses of the network to the applied input sequence. The simulation executes as though all bit values of each test-input vector are applied simultaneously, which is typical of a test fixture. This does not distinguish between pulse lines and levels and does not permit simple representation of time-staggered inputs as may occur when evaluating part of an assembly of gates.

The simulation proceeds as though all nodal activity reaches steady state before the next external input vector is applied. Thus, an external vector is applied (symbolically) by SIMUL to the nodes designated as external inputs. The output of each gate driven by an external input is computed to be a 0, 1, or X where X stands for don't know. All such gates are processed as one episodic event. A list is formed of those gates that are driven by a gate whose output has changed. When all input gates are processed, the change-list gates are processed, and a new change list is formed. Usually, logical activity will cease, and this is indicated when the next change list is empty. Continuous cycles are possible, and these may be automatically detected by the count of consecutive changes of each node.

The next input vector is applied when prior activity has ceased or a cycle has been detected. Each transition from old to new change list corresponds to a "unit" or "gate" delay. D4LASAR refers to this delay as 10 ns (nanoseconds), which is an arbitrary designation since the real gate delay could be any single value without affecting the meaning of the analysis. There is no provision for assignable delays, different gate delays (e.g., when both TTL and ECL devices are used) nor different rise and fall times. NAND-equivalent modeling in DLASAR assigns one delay per NAND, two delays per AND, OR, WIRED AND, and three delays per NOR. The only way to achieve the equivalent of assignable delays is to use multiple serial inverters, but this increases the running time and could cause the upper gate-count limit to be exceeded for DYSOGN, which uses SIMUL's model files.

SIMUL also provides checks on timing malfunction, both nominal and worst case. In worst-case delay analysis, convergent paths are examined for the consequences of one path at maximum accumulated delay and the other at least delay, where each gate has one delay plus or minus a selectable fraction. This includes the worst-case skew of the inputs. If one of these conditions could result in a latch being incorrectly set, then SIMUL will attempt to avoid the problem by adding "buffer" input vectors, which reduce the number of variables changing between successive inputs. It is possible to add enough buffer vectors to assure that only one input variable changes at a time. If the network has been designed to avoid races and hazards by clocking all latches, then the addition of buffer inputs should succeed. However, paths of unequal delay, which fan out and reconverge, cannot be compensated in all cases with buffered inputs. When "derace" is not successful, the affected outputs are marked with X as don't-knows and are disregarded thereafter.

SIMUL also has the capability to simulate single or multiple-fault conditions, but it simulates only one condition at a time. DYSOGN is much faster for fault simulation because it simulates many fault conditions simultaneously. However, only single "stuck-at" faults and shorts between adjacent pins of ICs or other equivalent modules are accounted in each condition.

In an ALEC-controlled iteration, REDUCE is used to discard unnecessary DYSOGN output. The final pass through REDUCE produces a failure dictionary. These data, along with the test sets, are stored on magnetic tape for use in testers.

For Tektronix S3260 testers, the taped data will permit ISO to provide automatic fault isolation using real failure data from the unit under test.

TECO permits the use of manually generated test sets and also provides macros for generating four different kinds of vector sequences. It also permits deletion, addition and rearranging of test-vector sequences. With appropriate job control cards, TECO activity can precede or follow ALEC iterations.

III. REQUIREMENTS FOR TESTING

An error is a result of incorrect performance. In this discussion, a fault is a hardware condition that is capable of producing errors. The existence of a fault in digital hardware is detected only by observation of the errors caused by the fault.

Failure Modes And Effects

Essentially all techniques for detection and diagnosis of faults in digital electronics depend upon analysis of the errors caused by faults. Infrared pattern monitoring, for example, has not proved to be sufficiently useful. Prediction of incipient failures through over-stress or marginal performance testing has never been sufficient for testing assemblies of solid-state digital devices. The design of automated fault detection and diagnostic procedures requires 1) prediction of the kinds of faults that may occur, 2) an estimate of their likelihood (some must be ignored), and 3) determination of the circuit behavior in the presence of each fault

Most test procedures are based on the assumptions that 1) once a fault occurs, it will persist until it is repaired, and 2) circuit behavior in the presence of each fault will be logical. The evidence published to date indicates that this latter assumption has been true for most faults in all state-of-the-art digital systems of the last, say, ten years. However, intermittent faults do occur, and some steady faults cause inconsistent errors (notably faults resulting in hazards or marginal timing). These (apparently) minority faults tend to require considerably more time to diagnose than the consistent majority because they require repetitive testing and will cause deterministic dictionaries to yield incorrect fault identification. The timing analysis of SIMUL can account for many of the timing problems under test conditions. However, D4LASAR provides nothing to help with intermittent errors due to chip defects, such as pinholes, marginal bridging between runs, undercutting or poor contacts. For these, the analyst may have to interact with D4LASAR and/or provide specialized test procedures.

Basic Requirements

1. The operation of the specimen machine must be definable in the absence of faults. (One cannot predict the consequence of a test performed on an unspecified black box.)

2. Sampled machine responses to test exercises must be consistently readable. (The machine must either halt to provide readout or must be synchronously sampled at an identifiable time.)

3. To differentiate any two faults (hence, to differentiate their locations), a test sequence must evoke a difference in at least one bit of the observable outputs when either one, but not both, of the faults exist.

4. Exercising and sampling a specimen machine must not cause a failure to occur. (E.g., testing shall not cause damage to circuits, nor cause erratic behavior due to loading, nor introduce significant noise.)

5. To compute the output sequence of a recursive function for an arbitrary input sequence, its internal state must be determined. (E.g., if the input to a J-K flipflop is JKC = 1 followed by C = 0, then the next output equals the last output. The next output is known only if the last output is known. Or consider the input sequence JKC = 1 followed by C = 0. The next output is changed if no fault exists but whether the output sequence is 0,1 or 1,0 depends upon the initial output value.)

Automated testing with predetermined input-vector sequences requires that specimen hardware containing memory be initialized to a predetermined state. Then a sequence of one or more preselected binary vectors is applied to the data and control inputs of the hardware. The hardware is exercised for a predetermined number of clock times; then the output is sampled. The observable results are compared with a precomputed set of values, which are the results expected of a fault-free machine. Observation of mismatch constitutes error detection. If no error is detected for a given test exercise, then either no fault existed during that test, or the test did not exercise the fault in a manner that would produce an observable error.

Typically a set of tests is selected with a capability for detecting a large fraction of the stuck-at-one and stuck-at-zero faults. It is preferable that each test meet the requirements of the foregoing paragraph (i.e., a set of independent tests). Some memory (flipflops, registers, RAMs and ROMs) will be directly accessible through normal operational data and control paths, and such memories may be tested independently of other hardware. Memory buried in logic must be initialized either through a "homing" sequence of inputs that will normally result in a selected initial state, or the memory is initialized directly through special test paths installed for that purpose. Provision for loading arbitrary initial values into recursive memory contributes significantly to shorter test sequences and easier selection of the test sequences.

Fault Location

Fault location has two requirements:

1. Identify which fault or fault set exists
2. Identify the module which contains the fault

The latter task (2) is a straightforward bookkeeping task after task (1) has been accomplished and fault sets have been correlated with physical layout.

Three different approaches to fault location are identified. The first is to provide a means of injecting test vectors and retrieving test results at the level of a replaceable unit. Thus, each replaceable unit could be tested independently of other units and its health ascertained from the test results Fault detection constitutes fault location in this case. This approach tends to require much

redundant hardware or requires the bed-of-nails approach to provide the necessary test access on boards. On whole chips it is just GC/NO-GO testing and is meaningless for fault isolation within the chip.

The second approach is a substitution technique. In its simplest form, a string of replaceable units is tested as a string. If an error is detected, each unit is replaced, one at a time, with an equivalent unit and the test is repeated. When the errors vanish, the last replaced unit is the suspect one. At the chip levels this approach requires a redundant reconfigurable chip design, which has been used for improving yield but not for on-chip fault location.

Guided-probe testing is essentially a variation of these two approaches when spare units are not available. A string of units may be tested as in the second approach. When a fault is detected, skip the last unit, monitor the output of the second last unit and repeat the test. If no error is detected the last unit is suspect; otherwise it is not, and the procedure may be repeated for the second to last unit, etc. Typically this approach is not desirable because the fault-free responses of each unit would be different, and because mechanized probing of the hardware is required. The latter requirement is likely to rule out dependence upon guided probing of LSI chips.

The third and most used fault-location technique utilizes error patterns for fault differentiatiation. To uniquely identify the existence of a particular fault it is necessary to execute a test sequence such that the response of the specimen machine with that fault will be different, in at least one bit, from the responses that would be obtained from the same test sequence with every other fault. In the stored-test approach, a directory is usually provided which lists the possible fault responses and identifies the corresponding faults that may cause each fault response. Correlation of test, fault, and response is determined by either programmed simulation of the faults or by insertion of single faults in operational equipment. Insertion of large numbers of single faults is impractical on chips, but fault simulation is entirely feasible.

The stored deterministic test approach to fault location is particularly appropriate when 1) the chip is modelled at the gate level and 2) the test-generation and fault simulation software is not overwhelmed by the chip complexity. When both of the conditions hold, D4LASAR will be very useful. If the second condition does not hold, then the chip function may be partitioned, and a combination of D4LASAR analysis and functional analysis may be used. If the first condition does not hold, then D4LASAR cannot be used, and surrogate techniques must be employed. Such alternative approaches are discussed in depth in Reference 7.

IV. D4LASAR FAILURE MODES

D4LASAR's DYSOGN module can simulate any one of the following three fault conditions:

1. Single stuck-at gate fault in real time
2. Single IC pin stuck at one or zero in real time
3. Single pair of adjacent IC pins shorted

In any one pass through DYSOGN, a multiplicity of single-fault conditions are simulated.

Figure 2 shows a two-input DTL NAND to illustrate stuck-at-zero (SAO) faults. The transistor could fail shorted from collector to emitter (SAO) or fail open. When driving a similar circuit with base pull up, the failed-open condition would be equivalent to an SA1. The condition of any one input diode

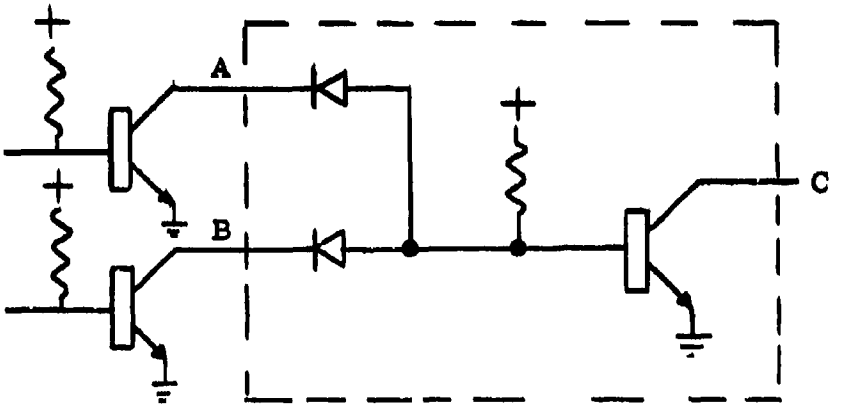


Figure 2. DTL NAND

open is functionally equivalent to that node SA1. Consider the four possible input combinations: AB = 11, 10, 01, 00. AB = 11 implies C = 0 so that C is tested for SA1. Either input open could not be detected, although either A SAO or B SAO could be detected. AB = 10 implies C = 1 so C SAO and B SA1 are detectable. AB = 01 makes A SA1 and C SAO detectable. AB = 00 detects C SAO but detects nothing that is not detected by other vectors. Thus, three input combinations constitute a necessary and sufficient test set for a single two-input NAND. For N inputs, N+1 combinations are required.

Appendix A offers an example showing how the major D4LASAR modules work with stuck-at failures.

V. STIMULUS GENERATION

Methods for selecting tests for digital networks fall in the following catagories:

1. Quasi-exhaustive
2. Pseudo-random
3. Functional
4. Deterministic
5. Nonsystematic (Manual)

The following discussion is primarily aimed at "random" logic, implying an exclusion of RAM and ROM, which are well ordered arrays Memory array testing is dominated by tests designed for detection of pattern sensitivity due to possible non-obvious internal cross-talk problems. The test designed for "random" arithmetic and control logic typically neglect pattern sensitivity. In some cases this may be a mistake, particularly with embedded RAMs and ROMs.

Quasi-Exhaustive Testing

If a device is entirely combinatorial, then it may be possible to generate all input combinations in a subjectively short time using a binary counter as the vector generator. Comparison of duplicate devices will obviate the need to compute the expected response for each vector, hence will permit full-speed test execution. Such exhaustive testing does not require fault simulation to prove that fault coverage is 100 percent.

If the device has recursive functions, then it is possible that necessary vector sequences will not occur in the counter-driven sequence. Fault simulation would be required to identify untested faults so the test sequence could be intelligently modified. The cost of fault simulation may significantly restrict the number of input variables that could be analyzed in exhaustive testing of recursive functions, as compared to the testing of strictly combinatorial functions.

The term quasi-exhaustive implies recognition of iterative structures in the logic that can be tested identically in parallel. Thus, the iterated parts are tested exhaustively in parallel, but all combinations of all input variables are not employed. Non-iterative functions such as carry lookahead will require specialized attention.

Pseudo-Random Testing

The intent and approach of pseudo-random testing is essentially the same as exhaustive testing, except that a small subset of all input combinations is employed. The results are checked either by comparison with the output of a duplicate process or with a computed (simulated) output. It is a Monte-Carlo approach that is blind to logic structure. Many careful analyses of fault coverage achieved by pseudo-random testing of specific logic networks have shown that it sometimes provides good coverage but usually does not. That is, the easily tested paths tend to be tested excessively while obscure paths tend to be missed. Lack of detailed structure precludes fault-coverage statistics.

Functional Testing

Functional testing is intended to limit test inputs to a set that exercises each specified system function once. The hope is that comprehensive fault coverage can be achieved with a nearly minimal test set, but without recourse to definition of actual logic structure. Where analyzed after the fact with full knowledge of the logic structure, seemingly complete functional testing usually has fallen far short of comprehensive fault coverage. This is typically due to 1) data-dependent alternative paths for individual functions and 2) the need for specialized vector sequences to test recursive functions.

The companion report (reference 7) provided by GEOS goes much more deeply into functional-test approaches.

Deterministic Testing

Deterministic testing begins with a definition of a set of failure modes, followed by selection of a test for each failure mode. The final number of tests is considerably smaller than the number of tested failure modes because signal paths are activated in parallel and multiple failure modes are detectable in the concatenation of circuits that compose each tested signal path.

Most automated procedures for testing digital devices, other than large memories, are designed to detect stuck-at faults. Fault isolation is obtained via redundant fault-detection tests (i.e., a test set that provides for multiple detections of the failure modes). Detection of shorted adjacent integrated circuit pins usually is accomplished with a test set derived for stuck-at faults.

Two requirements must be satisfied for detection of a fault: 1) external inputs and the internal memory variables must provide a local input to the failed device such that an output of the device will be erroneous when the fault exists, and 2) the device output error must propagate so as to be observable at an external output. The signal path from a detectable fault to an observable output usually will also permit detection of faults on intermediate devices Such a path is said to be "sensitive" (1) and may extend from external input to external output. In the presence of reconvergent fanout, a signal path may have sensitive and insensitive segments at the same time.

The best deterministic test results have been obtained with gate models of digital networks. The advantages of gate models derive from the fine structure of the modelling. The chief disadvantages are 1) long execution

time of automated test-generation and fault-simulation procedures and 2) increasing incidence of non-disclosure of integrated circuit gate models. Use of grosser models, such as a functional description of a MSI or LSI chip, tend to relieve these two disadvantages but incur the risk of inadequate fault coverage, increased likelihood of incorrect fault isolation, and inadequate timing analysis.

Nonsystematic Testing

Any test exercise, however obtained, will accomplish something. Given the existence of a non-void test set, the next test to be selected should contribute additonal fault coverage. This objective can be assured only if there exists a means of determining what has been achieved by the existing test set and what would be achieved by a proposed additional test. Such a means implies 1) the existence of a definition of failure modes, 2) a logical-network structure of sufficient detail and 3) a practical means of evaluating failure-mode coverage (e.g., a fault-simulation program). The implication is that nonsystematic nonexhaustive testing cannot offer assurance of adequate fault coverage.

For these networks containing a central processor, RAM, or other function, which is intended to assist the execution of the testing (i.e., a partially self-testing network), it often appears wise to test the simplest and shortest data and control paths first, then work into the more complex or obscure logical functions. Such intuitive approaches may offer an advantage such as improved ordering of tests, but their alleged advantages may be illusory and may prove unnecessary where systematic analytic approaches are employed.

Categories of Deterministic Test Generation Programs

A review of past and present deterministic test-generation programs is well beyond the scope of this report. However, there may be value in noting the following. All of these programs are intended to be, essentially, automatic and are most valuable for testing sequential functions. It is necessary to compute local functional (e.g., gate) values in both forward (input-to-output) and backward directions. Accounting for timing in a sequential machine gives rise to problems of significant difficulty. Most (perhaps all) test generation programs take advantage of the Huffman-Mealy model [2] by representing any digital sequential machine as an interconnection of two disjoint functions: 1) all memory elements and 2) all combinatorial logic. The approach taken, whether computing forward or backward, is to alternately, not simultaneously, compute the memory and combinatorial functional values.

Difficulties presented by timing representation include 1) nominal effects such as pulse representation, signal time order, nominal races and cycles and 2) worst-case effects such as unintentional pulse (spike) generation, races and hazards, and mal-ordering of signals.

The first test-generation programs provided zero-delay models without recognition of memory elements so the user had to identify all memory elements and the way they performed. These programs were put to good use, but they were inconvenient and sometimes yielded incorrect results due either to inadequate flip-flop models or lack of identification of memory elements created

by cross-coupled gates. Further, they offered time succession but no real time analysis.

DLASAR STIMGN represents a next generation where cross-coupled gate latches can be recognized automatically. STIMGN also can recognize and store, for later reference, the existence of logically inconsistent combinations (illegals) of memory element values. STIMGN has zero-delay gate models and, therefore, stumbles badly over real timing problems, but DLASAR provides more realistic unit-delay gate models in it simulator SIMUL. STIMGN requires a significant amount of workarounds -- special models to circumvent STIMGN's incorrect view of timing -- but experience has shown that the combination of STIMGN and SIMUL provides good timing analysis and effective test generation.

Elsewhere, work has been done on test generators that incorporate more realistic timing models with the intent to provide test generation, timing analysis, and fault-coverage analysis in a single integrated program. This is a very different task, and this writer is not aware that any such development has improved upon the STIMGN/SIMUL/DYSOGN capability.

STIMGN

Regardless of the primitives used in specifying the network model to INPUT, STIMGN sees an equivalent all-NAND model. If timing work-arounds are provided, then the STIMGN model will differ from the SIMUL-DYSOGN model. Conversion to all NANDs results in more gates and more gate delays than the user's model. The extra gate delays affect SIMUL but not STIMGN, since the latter ignores all delays. However, the added NANDS, increase size of the STIMGN files, increase STIMGN run time, and tend to make STIMGN's fault-coverage percentage optimistic. In every case, if the NAND-equivalent model is testable, then the user-specified model is equally testable, and with the same vectors.

Unless certain options are specified, STIMGN will begin with all nodes specified as don't-knows (X) and will select the first output listed on the OUTPUT control card. The subsequent STIMGN activity will cause that output to be sensitive to a string of possible faults. When not under ALEC control STIMGN will ignore all other outputs until the upstream logic is tested as completely as STIMGN can achieve. Under ALEC, STIMGN will go to the next output as soon as a test sequence is determined for the present output.

Starting with all nodes set to don't-know values, the source gate is initially assigned a zero value, which is backdriven to assign all ones to the gate inputs. Later the output will be assigned a one value, which will be backdriven to assign a single-zero vector to the input. The gate inputs are ordered by their driver (upstream gate) numbers in monotone increasing order. The assignment of a critical (defined later) input value starts with the lowest-order input and progresses to the highest-order input. Where possible, each gate will have input assignments of all ones and all single-zero combinations under circumstances where the possible gate-output errors would propagate to observable outputs.

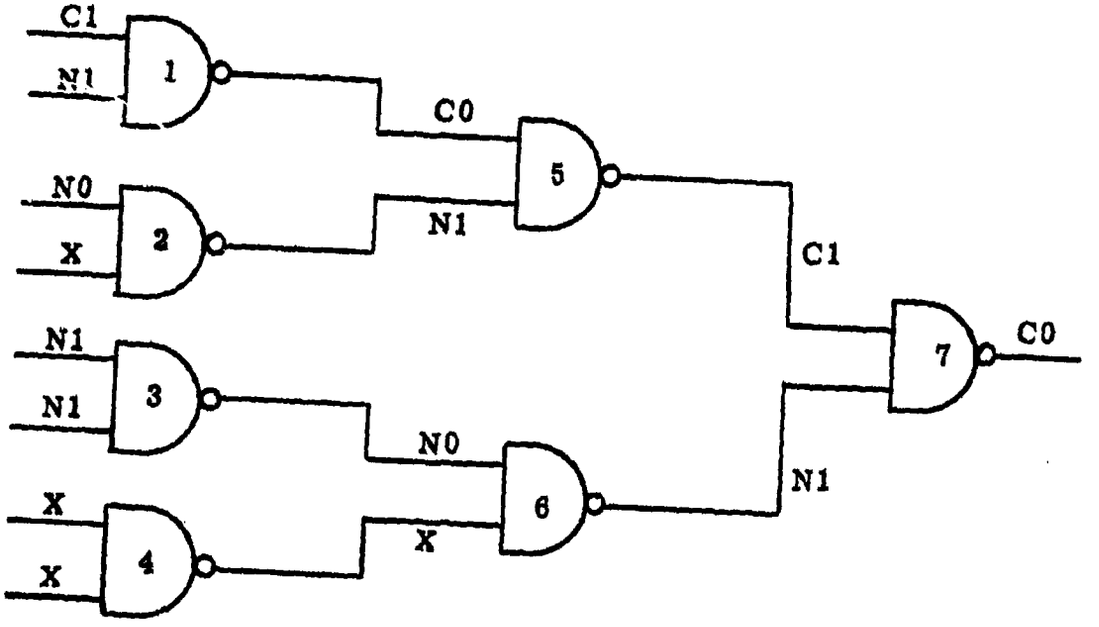
When any such input assignment is made, the next activity will be a forcing of the assigned values to directly connected gate pins to determine whether all logical relationships are consistent. If no logical contradiction is encountered (it will not be for the source gate but could be for other gates), then the next activity will execute. If a contradiction obtains, then that test attempt is aborted and a modification is attempted by "toggling" the last gate considered before the contradiction was computed.

"Toggle" means that the zero of a single-zero vector moves to the next pin on the same gate. Don't-know conditions which attended the toggled gate prior to backdriving that gate are temporarily restored. Thus, unsuccessful trial gate input vectors are discarded without trace except that they will not be repeated. If all toggle variations on that gate fail to resolve the contradiction, then the previous gate is toggled. This process continues until a noncontradictory condition is obtained or all toggling variations are exhausted. The latter conclusion means that the attempted gate test could not be achieved while using the designated output. If it had succeeded via a different output, STIMGN would not discover it until it eventually designated that output as a source and resumed its procedure or DYSOGN discovered it and reported it to STIMGN. This is one reason that STIMGN run time is reduced by ALEC through frequent use of SIMUL and DYSOGN to evaluate STIMGN results.

Assume the prior gate-input assignment did not result in contradiction during the forcing. For gates with critical zero (C0) output, the input is all ones and each input is critical. However, STIMGN is not designed to backdrive from all of these critical ones to the extent possible. Rather, STIMGN in effect selects a single one-valued input as critical while using a look-backward scheme to make a near-optimal choice. It then backdrives all of the remaining one-level inputs several (perhaps two) levels, but not to a greater extent. This obscure algorithm avoids exploring more than one critical path at a time, yet it appears to succeed in accounting for the inherent criticality of all of the all-one inputs. For gates with critical one output (C1), the backdriven input vector will always have a single zero, which will be critical (C0) and all other inputs will be ones which will be necessary (N1). For gates with a necessary zero (N0) output, the backdriven input vector will be all (N1). For gates with a necessary one (N1) output, the backdriven input vector will have a single necessary zero (N0) and all other inputs will be "don't-knows" (X). This helps to avoid unnecessary contradictions. See Figure 10 on page 15A.

The actual STIMGN details are not known, but presumably a contradiction between a backdriven input assignment and a prior driven input assignment would result in toggling the downstream input assignment that resulted in the backdrive. However, it is conceptually possible that the backdriven input would be toggled first.

As each such upstream assignment is made, its consequences would be forced, where fanout existed, to determine whether a contradiction would result. As before, contradictions result in toggling to attempt to find other noncontradictory assignments.



Note that D4LASAR has a "lookahead" algorithm which would probably assign a "1" for the X input of gate 6 and a "0" input for gate 4, thereby achieving tests of gates 3 and 6 even though they are not on a path marked critical. Gate 2 inputs would remain as N0 and X.

Figure 10. STIMGN Example

This process continues until all of the next upstream nodes are external inputs and all the latches have zero inputs. If this condition is satisfied, then a particular test has been completed, and STIMGN will identify as detected the failure modes corresponding to the string of critical input vectors. If at least one latch having all-one inputs is interposed, then the latch value is recursive, and the latch and external input values so identified provide one total state to be achieved in a test sequence. The recursion makes it necessary to find the total state that must occur earlier in actual testing to achieve the required present set of latch values. Thus, the process drives backwards through the latches and continues as before.

As described, STIMGN generates a single string of critical ones and zeros for each test sequence. By construction, each of the critical input vectors tests a failure mode STIMGN can determine, and these are listed as detections. Where reconvergent fanout exists, some of the apparent detects will not be valid. (See Appendix B.) This determination and that of identifying incidental parallel detects is a task left to DYSOGN. Timing analysis is left to SIMUL. Compaction of tests sets is left to OVRLAY.

The toggle scheme is used for more than avoidance of logical inconsistencies. Assume STIMGN is not executing under ALEC so activity remains associated with just one source output. When a test set is completed, the last active gate is toggled to initiate a new test set. Forcing, testing for inconsistencies, and further toggling continues until a complete new test is selected. This procedure determines the necessary initial conditions for the new test and avoids the need for recalculating a critical path because that previous path is used to the extent possible. Further, the toggle scheme simplifies the problem of avoiding repetition of previous tests.

When operating under ALEC, STIMGN changes source outputs after each test set determination. The status of the previous critical path and associated value assignments are stored so when STIMGN returns to a source output, it resumes as though there had been no interruption other than the increased list of detections.

Illegals

If a combination of latch values is sought but found to be contradictory, then that combination is labelled "Illegal". Illegals are filed for later reference to avoid futile backtracking. The program WHITLE merges the logical combinations, which identify Illegals, to produce more compact definitions of the Illegals.

There are two Illegal tables, permanent and temporary. If the latch values comprising an Illegal are initially "don't-knows", so all possible combinations are explored, then the Illegal goes to the permanent-Illegal table and is held until the job is ended. If latches comprising an Illegal are conditioned by prior critical or necessary assignments, then the Illegal goes into the temporary Illegal table to be held only as long as the conditions remain valid. "Illegal PREPROCESS" is an optimal STIMGN procedure that starts with latches closest to external inputs and tries to initialize each latch in all possible ways. The goal is to preload the permanent Illegal table which

reduces the eventual load on WHITLE and STIMGN.

WHITLE

If a combination of latch values is sought by STIMGN but found to be contradictory, then that combination is identified as "Illegal" and filed to avoid future futile backtracking. The purpose of WHITLE is to combine the Illegal combinations to reduce the file space.

WHITLE details have not been disclosed; WHITLE could work as follows: If each latch were uniquely labeled, e.g., A or $\bar{A}$, according to whether the latch value is one or zero for a particular Illegal, then a concatenation of appropriate labels would designate an Illegal intersection of latch values. For example, $A\bar{B}C$ could identify an Illegal that exists when latches A and C are one-valued while B is zero-valued. The list of Illegals forms a union of latch-value intersections, which corresponds one-one to a Boolean function in sum-of-products form. Thus, the problem of reducing the list of Illegals is equivalent to the classical problem of minimizing a Boolean sum of products. Near-minimal reduction will be satisfactory for Illegal lists.

Two kinds of redundancy in Boolean sums-of-products formulas are "literal" and "term". Use of the following identities will provide a near minimal reduction.

Let A, B, C ... be the Boolean variables representing a set of latches. Symbols + and · will represent Boolean operators sum and product respectively.

$$A+\bar{A} = 1 \qquad A\cdot\bar{A} = 0$$

$$A+A\cdot B = A, \qquad A+\bar{A}\cdot B = A+B$$

$$A\cdot(A+B) = A, \qquad A\cdot(\bar{A}+B) = A\cdot B$$

$$A\cdot B+\bar{A}\cdot C+B\cdot C = A\cdot B+\bar{A}\cdot C$$

The latter identity is discovered by Mott's[3] expansion theorem whereby a set of terms is formed in the following way: For each pair of terms in the sum-of-products formula, which have a literal complemented in one term but not in the other, form the product of the literal of both terms excluding the singly complemented term. Thus, for $(A\cdot B + \bar{A}\cdot C)$, form the product $B\cdot C$. If that product is not zero, then join that product to the union of redundant terms. When all such pairs have been processed, cancel those terms in the original formula which appear in the redundant set.

Example

CD \ AB	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	1	1	1	0
10	0	0	1	0

$$F = BD + \bar{A}\,B\,\bar{C} + A\,\bar{C}\,D + A\,B\,C + \bar{A}\,C\,D$$

$\bar{A}B\bar{C} + A\bar{C}D \longrightarrow B\bar{C}D$ $\longrightarrow$ = "implies"

$\bar{A}B\bar{C} + ABC \longrightarrow 0$

$\bar{A}B\bar{C} + \bar{A}CD \longrightarrow \bar{A}BD$

$A\bar{C}D + ABC \longrightarrow ABD$

$A\bar{C}D + \bar{A}CD \longrightarrow 0$

$ABC + \bar{A}CD \longrightarrow BCD$

$B\bar{C}D + BCD \longrightarrow BD$

$\bar{A}BD + ABD \longrightarrow BD$

hence BD is redundant in the F formula.

OVRLAY

OVRLAY is designed to reduce the number of STIMGN-generated test sets. It takes advantage of don't-care inputs by assigning them, where possible, to allow two test sets to be merged into one.

Consider two test sets labelled R and S where $R \leq S$ (i.e., with respect to numbers of patterns). OVRLAY compares the i^{th} pattern of R with the i^{th} pattern of S for $i = 1, 2, \ldots, i_{max}(R)$. If no bit of R is the complement of the corresponding bit of S for all i, then R and S are replaced by S' where the zeros and ones of R replace the corresponding don't-cares in S and vice versa.

If at least one bit of R and S are complements, then set R is realigned one pattern position with respect to S ($i+1$ pattern of S compared to i^{th} pattern of R for all i) and the process is repeated. If no merging of R and S proves possible, then each is processed similarly with the other test sets.

It is not known whether the R, S set comparison is stopped when the last patterns of each is aligned. Presumably this is true; otherwise it would be possible for appropriate test sets to merge the first pattern of R with the last pattern of S to form a merged set of R+S-1 patterns.

When OVRLAY has completed its merging procedure, it assigns each remaining don't-care to the previous assigned value. This minimizes the number of variable changes due to don't-care assignments to help reduce the incidence of races and hazards, virtually eliminating any possibility of further merging in the event that OVRLAY were reentered (which would require special user control).

Note that OVRLAY does not attempt to reduce the length of test patterns. Also, it precedes the execution of DERACE in SIMUL, hence it has no opportunity to merge patterns after DERACE has added buffer patterns unless SIMUL patterns are resubmitted to OVRLAY, which requires special control. Further, under ALEC, OVRLAY operates only on the STIMGN patterns of each ALEC pass; hence OVRLAY is not permitted, under ALEC to provide a near-optimal merge. Incidentally, OVRLAY is not likely to achieve, under any available control scheme, the best possible merging because each merging of two test sets constrains possible further merging with other test sets.

VI. LOGIC AND TIMING SIMULATION

Introduction

When given a digital network model and a sequence of stimuli, logic simulators compute the fault-free response of the network to the stimuli. There can be a considerable variation in the representation of gate behavior depending on what the simulator designer wished to achieve. Most simulators provide three states per node to represent 0, 1, and don't know. The simulator in reference 4 provided 0, 1, and two don't know states, one non-propagating for initializing the simulation and one propagating. Eichelberger[5] showed how three states may be used to represent static one, static zero, and transition. This provides a means of analyzing hazards and races, and is incorporated in the Hewlett-Packard TESTAID-III simulator. Additional states may be employed to represent the high impedance of tri-state bus drivers, and signal rise or fall. Adding states provides improved realism but also additional execution time.

Modelling of sequential activitiy requires a representation of time. Some early simulators were based on the synchronous design of the logic in that combinational logic was simulated without implied circuit delay, while flip-flops were defined by the user and allowed to change state only under control of an explicit clock. This did not handle unclocked latches and did not account properly for unidentified memory elements.

Unit gate delay implies that all gates have the same delay. Typically the bits of each input stimulus are applied to the network simultaneously, then the outputs of the first level of gates are computed as one event. Where the output values have changed and drive downstream gates, those gates are put into a change list. When the present level of gates has been completely processed, then the gates in the change list are similarly processed so that a new change list is formed. This process continues until activity stops, as indicated by an empty change list, or a run-time threshold has been exceeded (e.g., a very long counting sequence). Each pass through a change list constitutes an epoch that marks one interval of an implicit clock. The unit-delay representation does not require identification of explicit clocks and accounts for memory-element behavior without help from the user.

A need for more realistic modelling of timing arises where different solid-state technologies, e.g., TTL and ECL, are used together, and where custom high-performance chip designs are to be modelled; i.e., where circuit timing is adjusted by the designer to meet the performance requirements. Variable gate delays may be obtained by making the unit gate delay equal to the greatest common denominator of the delays to be used in modelling the real circuits. Then single-input gates are concatenated to add the necessary delay to each decision gate. The consequences of the increased gate count may be intolerable: more memory will be required by the simulator and the execution time will be slower by a factor equal to one plus the number of dummy gates required to compose the average simulated gate delay.

A better alternative for achieving assignable gate delays is to use the so-called next-event timing. Each gate type requires a multi-bit time tag, which is loaded at model-generation time with the appropriate number of greatest-common denominator time delays. Gates of the same function, e.g., three-input NAND, may be defined with different delays by assigning suitably different labels and the appropriate time tags. When any gate is processed and its output is found changed, the immediate down-stream gates are put in a change list similar to that of unit delay processing except that the time tags will be in monotonically increasing order. The next gate to be processed will be at the head of the list and will have the smallest (not necessarily unique) unprocessed time tag. Each different time-tag number corresponds to a gate activity. A savings in processing time is achieved by skipping over intermediate unrepresented time tags.

SIMUL

The logic simulator for D4LASAR is SIMUL, and it is a three-state unit-delay simulator. A variation that permits assignable delays has been developed but has not been released because it is seven times slower than the unit-delay SIMUL.

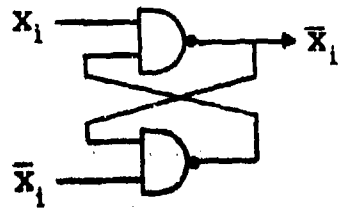
SIMUL accepts stimulus patterns from OVRLAY, TECO or (user-specified) PATGEN and drives these forward to determine their effects on all nodes for all patterns. Initially all network nodes are set to the don't-know state, X. The first pattern is applied to the gates connected to the external inputs, and the gate outputs are calculated. This constitutes time "0". The gates with changed output are listed in a status table; the gates driven by the changed outputs are listed, and their outputs are calculated. This constitutes time "1". The new changes are incorporated in the status table, and the process continues. The signal changes propagate through the network like a wavefront, one level of gates at a time. There is no processing of static gates, which is a significant time saver. This procedure continues until the next change list is empty. Then all of the nodal values are saved in a file (for either SIMUL printout or later use by DYSOGN), and the next input stimulus is applied.

Known Relationships Between Unknowns

There are circuits that are well-behaved but whose output cannot be computed when all nodes are initially don't-knows without other information. If a NAND gate has a zero input, its output will be one. With a single don't-know input X and all other inputs one-valued, the output is $\bar{X}$.

For $\bar{X}$ to have meaning, the X must be subscripted to identify which node is X and which is $\bar{X}$. Both X_i and $\bar{X}_i$ into a NAND is equivalent to a zero input which causes output of one, regardless of other inputs. The output of a NAND loses its subscript information if the inputs are X_i, X_{j+1}, 1, . . . , 1, unless X_i and X_j are strongly related in a known way.

A necessary relation for logic simulation is that of a latch with complementary don't-know inputs.



DLASAR retains these relationships in its logic simulation.

The use of strictly NAND modelling simplifies the simulation procedure and helps to speed the process. However, an all-NAND representation contributes unreal timing since only NANDs have single unit delays. AND and OR gates require two levels of NANDs and, therefore, incur two unit delays. WIRED AND (incorrectly named WIRED OR in DLASAR documentation) incurs two delays unless modelled by direct connections of no delays. NOR gates require three levels of NANDs and three unit delays. The lack of assignable delays yields unrealistic delays and possible incorrect race/hazard analysis when primitive functions other than NAND are used.

The irrevocable procedure whereby only one external stimulus is applied while the network is active is appropriate for test fixture applications where stimuli are held static until the unit under test becomes static. However, it gives rise to irksome or serious problems where logic is clocked, or is pipelined, or is subject to time-staggered inputs.

Clocked Logic

When a device is carefully designed for clocked operation to avoid races, hazards, and cycles, the clock input will be treated by SIMUL (and most other simulators) as though it were an ordinary data input. The logic designer's timing rules are frequently violated for latches (data inputs should be static before, during, and after the clock comes on) so races and hazards result. SIMUL will spend much time analyzing the races and hazards and will attempt to add "buffer" stimuli to reduce the number of inputs changing at any one time. It is possible to use the editing capability of TECO to force stimulus bit changes in the manner of a clock, but the procedure is not automated -- clock input cannot be specified -- so that tedious scanning of and addition to OVRLAY stimuli is required.

The choices available to the user for testing well designed logic are 1) allow race/hazard analysis to execute (by default) and accept the resulting increased SIMUL run time, enlarged buffered data sets, and warning messages, or 2) shut off the race/hazard analysis (option) and incur races and hazards because the stimuli are violating design rules relating to clocking, or 3) manipulate the OVRLAY stimulus sequences via TECO such that the model executes as the designer intended with respect to timing. It would be desirable to have an option that permitted an input to be identified as a clock, with the

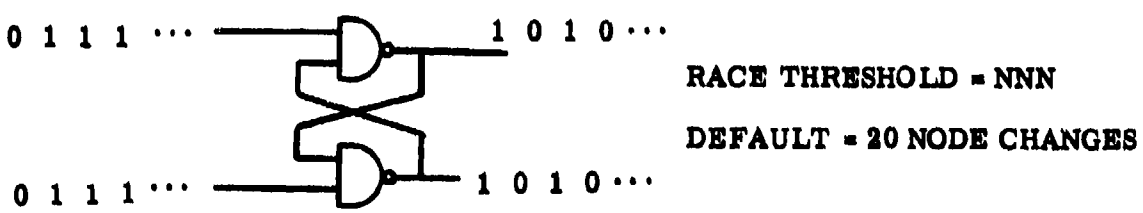
property that it would be automatically specified in SIMUL, prior to race/hazard analysis, as off when data and control inputs were changing, and as on when the other inputs were static.

Race/Hazard Analysis

If the inputs to a two-input NAND change from 0, 1 to 1, 0 (or vice versa) then, depending on their relative timing due to upstream delays, they may have momentary values of 0, 0 or 1, 1. The 0, 1;0, 0;1, 0 sequence will assure a constant NAND output of 1. The 0, 1;1, 1;1, 0 sequence will cause the NAND output to change from 0 to 1 and back to 0. If momentary, this pulse is referred to as a spike. It may be intentional as with certain designs of pulse generators, or it may be an unintentional consequence of circuit delays. Whether the end result of an unintentional pulse is serious depends on the duration of the pulse and whether it affects a latch. In the switching-theory literature, such an effect is called a race if the spike-producing variables are due to memory elements switching at different times, and the effect is called a hazard if combinatorial circuit delays produce the delay differences of two otherwise simultaneously switched variables. A race or hazard is critical if it causes a latch to be set incorrectly. DLASAR's SIMUL evaluates both races hazards, and, generally, reacts only when critical. SIMUL warning messages use terminology different than that of switching theory literature.

Nominal Race

Timing analysis is done for both nominal and worst-case delay distributions. The warning message "nominal race" can be caused by two different circumstances. If, as shown in Figure 3, a latch has zero-valued inputs followed by all-one inputs for successive change-table times, then a classical static hazard will occur because the outputs should, according to normal latch operation, remain at the previous value. They will be indeterminate because the latch could settle either way. The change-table simulation of SIMUL will calculate alternate 0, 0 and 1, 1 latch-output pairs. SIMUL will monitor the number of times each node changes for each input vector. If the change count for any node equals the race threshold, the warning NOMINAL RACE will be declared and the node identified for that input vector. "RACE THRESHOLD" is a SIMUL option that may be set as high as 999 and which defaults to 20.

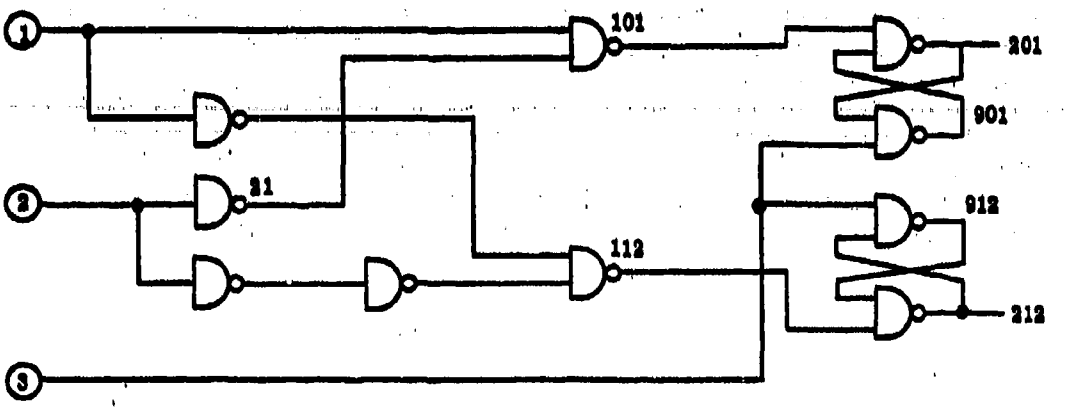


This "nominal race" is the classical "static hazard" of set-reset flip-flops where the outputs should be 1XXX···but are simulated by SIMUL as shown. After the race threshold is equalled, the latch outputs are labelled X.

Figure 3. Nominal Race (type 1)

Static hazards are thus identified as nominal races. It is possible to trigger this alarm without a latch. For example, a simulation of a new 16-by-16 combinatorial multiplier design resulted in a declared nominal race for RACE THRESHOLD = 20 (default value). This phenomenon occurred because the logic chains were very long and were impacted by other logic chains at many nodes. Thus, some gates happened to change outputs repeatedly before settling into their final values, and the change counts exceeded the threshold. Since no timing problem really existed, the RACE THRESHOLD was increased so the number of gate changes was then effectively ignored.

There is a second way to obtain a warning of NOMINAL RACE. If one latch input has a value of one and a negative pulse of one-gate-delay duration drives the other latch input under nominal conditions, then NOMINAL RACE will be declared and the appropriate location information given. Figure 4 shows such a condition.



Input Vector 1: 1,2,3 = 1,1,0; 201,901,212,912 = 0,1,0,1
2: 1,2,3 = 0,0,1; = 0,1,X,X

Nominal Race Message: Nominal race on pattern 2 found at node 212. The following additional nodes will be set to X -- 912.

Possible Spike Message: Possible spike at node 101 due to 21 and 1 has a nominal margin of 10 nanoseconds.

Figure 4. Nominal Race (type 2) and Possible Spike

Possible Spike

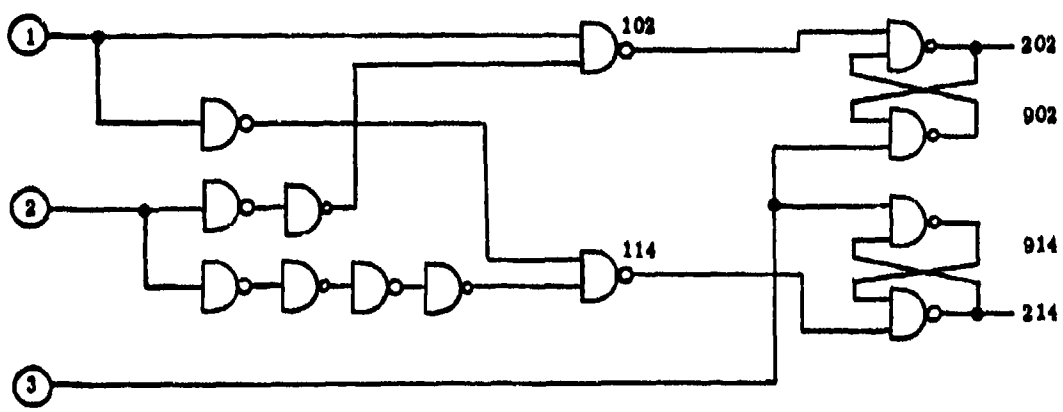
If under normal conditions a pulse is generated at the input to a latch which under ideal conditions will not change state but might under non-ideal conditions be affected by the pulse, then POSSIBLE SPIKE is declared with the spike width and location information. Figure 4 illustrates an example of a possible spike. Note that 10 nanoseconds is one gate delay in DLASAR messages. Had the delay difference been three gates (it must be odd), the message would have stated 30 nanoseconds.

Worst-Case Race Analysis

In addition to nominal races, races may occur due to transient conditions which derive from worst-case delay conditions. The analysis is done as follows.

After completing the forward-drive simulation and the nominal-race analysis, SIMUL back-drives with nominal delays and looks for gates whose present and prior output values were both one. If, in such a case, two of the gate inputs had changed in opposite directions, then a forward trace is executed to determine whether a latch would be affected. There is no analysis to see whether that changed latch value would have any significant further effect, and in that sense the SIMUL analysis is pessimistic.

If there were a spike and a latch apparently affected, then a back trace executes with worst-case delays in the two paths that cause the spike. If there is a nominal delay difference, then it is, in effect, assumed that the designer wanted one path slower than the other. Worst case is defined as a condition that tends to reduce delay in the longer path while increasing delay in the shorter path. This is accomplished by subtracting the TOLERANCE percentage of a single gate delay from each gate in the longer path and adding the same percentage to each gate in the shorter path. If the worst-case pulse width is less than the "SAFE TIMING" option value, then the warning "ASYNCHRONOUSLY SET LATCH" is produced. (See Figure 5.) Otherwise, the condition is assumed to be satisfactory and no message appears. The SAFE TIMING threshold may range from zero to 99.9 gate delays. Default is 10 gate delays.



Input Vector 1: 1,2,3 = 1,1,0 ; 202, 902, 214, 914 = 1,1,0,1
2: = 0,0,1 ; = 0,1,1,0

Asynchronously Set Latch Message: Latch nodes 214 and 914 asynchronously set to 1/0 due to an unsafe 30-nanosecond pulse on node 114.

Dangerous Timing Message: Latch nodes 902 and 202 were at 1/1 after the last pattern and are at 0/1 after this pattern due to 3 going to 1 on level 5 and 102 going to one on level 6. Timing margin = 10 nanoseconds.

Figure 5. Asynchronously Set Latch and Dangerous Timing (SIMUL options: NO DERACE, SKEW = 0, otherwise default)

Figure 5 illustrates one other worst-case timing circumstance analyzed by SIMUL. If the inputs to a latch change from 0, 0 to 1, 1 and the nominal output is deterministic (i.e., the nominal input delays assure no nominal race), but worst-case delays could change the times that the inputs switch, such that the opposite latch value would obtain, then the message "DANGEROUS TIMING" would be produced.

Race Ratio, Path Ratio, and Tolerance

The worst-case race analysis assumes that the nominal ratio of total path delays is correct and that a timing problem would exist if the path with fewer gates had a delay equal to or greater than the path with more gates. SIMUL assumes a nominal unit delay per gate and, in worst case, a gate delay of $1 \pm T$ where T = TOLERANCE/100. TOLERANCE is the user-specified option which is the fraction of a unit delay by which any gate can deviate from nominal in worst-case conditions. The default value for TOLERANCE is 33 (percent) for which worst-case gate delay is 0.67 or 1.33.

A race exists for a converging pair of paths of gate counts N_{SHORT} and N_{LONG} if a latch is affected and:

$$N_{SHORT}(1+T)/N_{LONG}(1-T) \geq 1$$

Define race ratio $R = (1+T)/(1-T)$

$$\text{path ratio} = N_{LONG}/N_{SHORT}$$

Then a race exists if

$$[N_{SHORT}/N_{LONG}] \cdot [R] \geq 1$$

or $$N_{LONG}/N_{SHORT} \leq R$$

No race if $N_{LONG}/N_{SHORT} > R$

This is summarized in Figure 6 where race ratio is plotted as a function of T. For example with default TOLERANCE = 33, T = 0.33 and R = 2.0, so any path ratio greater than two cannot cause a race, while a ratio less than or equal to two can cause a race. This curve can be used by designers to establish path ratios for a given T, and to simplify visual checks of schematics for possible races. It also suggests that SIMUL's race analysis uses a simple arithmetic process once the suspect path pairs have been identified and the gate counts established.

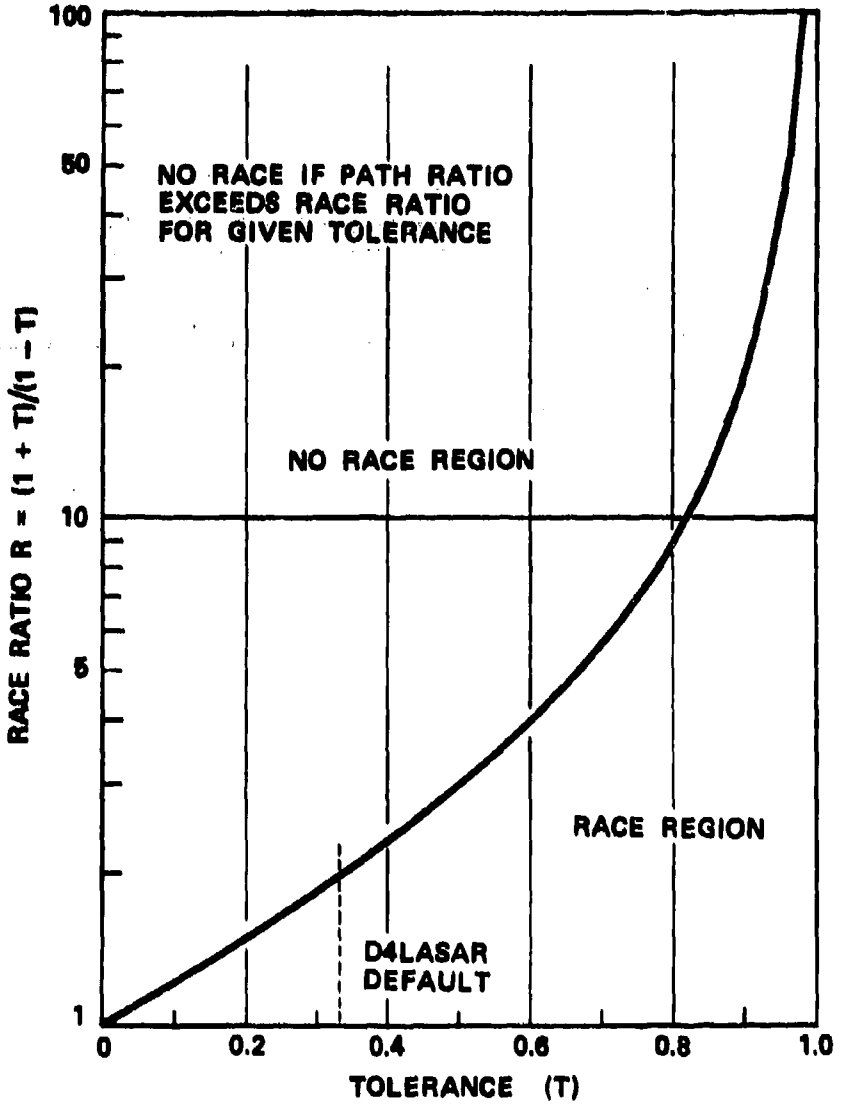


Figure 6. Race Ratio versus Tolerance

SKEW

A SIMUL option is SKEW, which assigns a delay variance to the hypothetical tester that provides the input vectors. SKEW ranges from zero (simultaneously applied inputs) to 999.9 gate delays. Default is 200 gate delays. During worst-case race analysis, SIMUL takes the worst of lesser/greater delays for each path pair that could cause a race. SKEW cannot be used to assign fixed delays to provide time-staggered inputs because SKEW provides both lesser and greater delays. There is no option to provide inputs with fixed time stagger.

DERACE

If the "NO DERACE" option is not specified, then SIMUL will attempt to remove race conditions by adding "buffer" vectors to the test-vector sequence in such a way that the number of external input bits that change at any one time are reduced. Buffer vectors are labelled. Where reconvergent fanout is causing a race, buffer vectors cannot resolve the race so none are added for such races. Instead, the appropriate warning and location information are produced, and any affected output is marked "X" so it will not be used by DYSOGN. There is a "NO X" option in the event the user chooses to disregard the consequences of races discovered by SIMUL.

Commentary on SIMUL Timing Analysis

Specification of a single nominal delay per gate and a single worst-case delay tolerance does succeed in discovering many timing problems that are shown to be real when the circuit is examined. However, real-world gates vary considerably in range of delay, particularly when both slow and high-speed technologies are used together (e.g., low-power Schottky and emitter-coupled logic). In such circumstances assignable gate delays are desirable. This is particularly valuable when simulating circuits with significant time stagger in the input signals. Digitest has developed a "variable delay" option but has not released it because it incurs a slowdown by a factor of seven. Some increase in run time appears inevitable with assignable delays because there must be increased file processing. However, it is not clear that the run time penalty need be excessive if an appropriate algorithm is used.

The sum of worst-case gate delays along a signal path becomes increasingly pessimistic as the path length increases. It is by no means clear as to how to interpret or circumvent this problem. The unit delay per gate is an abstraction not subject to change. However, a race derived from relatively long chains of gates tends to have an exaggerated estimate of the delay difference. Such races could be re-evaluated with TOLERANCE set to a smaller value, which would reduce the worst-case delay difference.

There are other contributions to inaccuracy in timing representation. Differences in rise and fall time will tend to average out. Nominal delays not centered between minimum and maximum worst-case delays are not likely to contribute serious timing errors. However, conversion of other primitives to NANDs can result in significantly different timing representation, and there is no simple way to counter this in D4LASAR.

To summarize SIMUL-timing comments, timing representation may be significantly erroneous, yet the results of the SIMUL race/hazard analysis are usually useful. Understanding of the timing representation is necessary because:

1. Race/hazard options must be chosen. The default values may be incorrect for the circuit modelled.

2. Partitioning of a large network into manageable subnets may require a special timing specification.

3. The reason for race/hazard warning messages must be determined so a decision can be made to ignore them or prevent the race.

VII. FAULT SIMULATION

Introduction

Where the basic purpose of a logic simulator, such as SIMUL, is to compute the expected responses of a fault-free network to the excitation of a sequence of input vectors and memory states, the basic purpose of a fault simulator is to do the same with all variations of the network under all the fault conditions of a specified fault class. This leads to the most important problem of fault-simulation programs, namely the large number of fault simulations required and the total execution time.

Roughly 3.5 stuck-at faults per gate are possible for a typical network; hence, for a 20,000-gate processor with at most one fault at a time, about 70,000 faults require simulation. In a production environment, or in a field environment where repair is delayed, there may be more than one fault at a time. If the 70,000 faults were accounted two at a time, then over 2 billion single and double fault conditions would require simulation. Hence, multiple fault conditions are rarely considered. Some studies reported in the literature have shown that tests that achieve 100 percent single stuck-at fault coverage will also detect most of the multiple stuck-at fault conditions. Other studies also indicate that tests that provide high-fault coverage of single stuck-at faults will also provide high coverage of single bridging faults (that is, shorts between adjacent signal paths). It should be noted that a fault directory, based on detection of single stuck-at faults, may provide excellent location information for single stuck-at faults, but is likely to provide incorrect location information for bridging or multiple stuck-at faults.

What about other fault types such as inadequate timing margins? When a simulation program is likely to require long run times, one does not wish to burden it with further tasks such as increased size of fault set or analysis of timing malfunction. Consequently, D4LASAR is designed to execute most of the race/hazard timing analysis in SIMUL rather than DYSOGN. DYSOGN provides analysis for nominal races, but no other analysis of fault-derived timing malperformance. The risk incurred by omission of fault-derived worst-case timing analysis is not known.

The time to complete fault simulation is reduced primarily through some form of concurrent processing. "Parallel" fault simulators execute multiple simultaneous simulations. Three possible parallel fault-simulation approaches are:

1) Assign outputs of different gates to the bits of a word and apply the inputs, corresponding to execution of one test at a time, to these parallel-processed gates. This approach is facilitated if the gate functions are identical. An n-bit word would account for at most n failed gates.

2) Assign the output values of a gate, which correspond to different failure modes of that gate, to different bits of a word. Then a single test will execute at a time with inputs to each gate appropriate to the input combinations required to evoke an error if the corresponding failure mode existed. An n-bit word would account for, at most, n failure modes per gate.

3) Assign a single gate to each word with each bit modelling the output value of the gate for a unique test. An n-bit word would account for at most n simultaneous tests.

Parallel fault simulators have been superseded by fault-list simulators based on a scheme of Armstrong's.[1] Where parallel fault simulators require multiple passes through the network model while simulating n (n = number of bits in word) faults at a time, the fault-list simulators account for all of the faults concurrently as the simulation processing proceeds from inputs to outputs for each test. The fault-list simulators are faster than the parallel simulators for large networks because the time saved by avoiding iterative simulation is greater than the time lost in processing the very large lists of faults. For a range of "typical" networks, the run time of parallel fault simulators increases roughly as the number of gates to the 2.5 power, while the run times of fault-list simulators increase roughly as the number of gates to the 1.5 power.

The following description of fault-list simulation is based on reference 4. DYSOGN uses a similar technique but with added features that have been found to provide significantly faster processing.

Fault-List Simulation

A fault list is a set of faults with the following properties: 1) a fault list is defined for each digital-network node for each time it is in steady state, and 2) the list for each gate identifies all of the stuck-at* faults that could be detected if that gate were observable. The fault-list algorithms account for single and multiple faults and are constructed to account for a gate fault propagating through the same gate during later iterations.

Reference 4 was the first published algorithm for fault-list simulation. Unfortunately, the equations contain an error. The equations here are an adaptation of the fault-list portion of that report. The description here will first cover fault-list simulation where there are no don't-know conditions, then these will be accounted afterwards.

Let A and B be any two fault lists. We define four operators, three of which are set-algebra operators.

Union $A \cup B$ contains those faults contained in A or B.

Intersection $A \cap B$ contains only those faults common to both A and B.

---

* Outputs stuck at one or zero and inputs stuck open. When an input is open, the gate sees a stuck-at one input value. Input values stuck at zero can only be due to the upstream gate output being stuck at zero when fanout is unity, under the assumptions used here.

Difference A-B contains those faults of A which are not in B. ($A-B = A \cap \bar{B}$)

Exclusion $A \theta B$ contains the faults of A if and only if B is empty.

The use of the fault-list difference arises from the need to identify the case where two inputs are switched from (0, 1) to (1, 0) by a fault. Such input faults will not be transmitted to the gate output fault list.

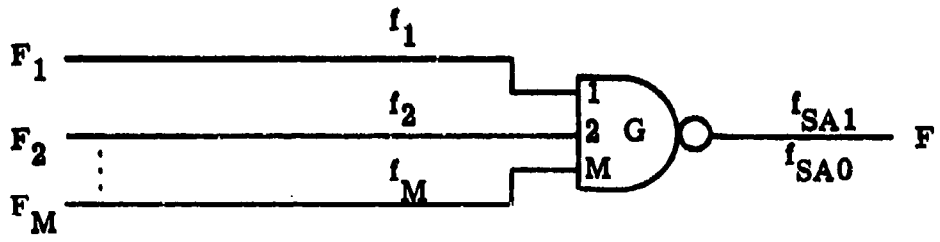
The use of the fault-list exclusion arises where a node fails to its normal value. While the nodal value does not change, there can be no fault detection at that node and no upstream fault list can be transmitted through that node.

Difference precludes some of the input fault list while exclusion precludes all of the input fault list where the appropriate conditions hold.

Fault lists may be defined on any primitive function, but will be defined here only for a generalized NAND gate. Figure 7 diagrams a NAND with m-many inputs, establishes the required labels, and presents equations for computing the output fault list as a function of 1) the fault-free input vector, 2) the fault lists associated with the sources of the inputs, and 3) the faults which could be contributed by the gate alone. Inputs are numbered 1 through M. The fault lists associated with the input sources are labelled F_1 through F_M. Inputs open are labelled f_1 through f_M (each open input would be equivalent to a stuck-at-one value on that input). Output stuck at one or zero is labelled f_{SA1} and f_{SAO} respectively. The final output fault list is labelled F. Inputs stuck at zero are not considered primarily because most gate implementations do not have a significant corresponding failure mode.

When more than a single input is zero or one-valued, it is convenient to handle them in sets. The symbol $\bigcup^{k}$ designates the union of fault lists on those inputs that have the value k, where k is zero or one. The symbol $\bigcap$ is defined similarly for the intersection of fault lists.

Equation 1 in Figure 7 is identical to Equation 1 of reference 4, page 1464, except that reference 4 does not recognize the need for both difference and exclusion operators. Equation 2 of reference 4 is subsumed by Equation 3 and is not included here. Equation 2 of Figure 7 differs from Equation 3 of reference 4 in that the set difference (rather than exclusion) is taken between the intersection set and the union set and exclusion, not difference, is used elsewhere. Equation 4 in reference 4 will be discussed later.



F_i = Fault list on Gate Driving i^{th} Input of G

f_i = i^{th} Input Open

f_{SAk} = G Output Stuck At k, k = 0, 1

F = Fault List on G Output

$\bigcup^{k}$ = Union of Fault Lists on Inputs Equal to k, k = 0, 1

$\bigcap^{k}$ = Intersection of Fault Lists on Inputs Equal to k, k = 0, 1

1. All Inputs = 1, Output = 0, G = NAND

$$F = \left[\left|\bigcup_{i}^{1}(F_i \,\theta f_i)\right| \bigcup f_{SA1}\right] \theta\, f_{SA0}$$

2. Set i Inputs = 0, set j Inputs = 1, Output = 1

$$F = \left[\left|\left[\bigcap_{i}^{0}(F_i \cup f_i)\right] - \left[\bigcup_{j}^{1}(F_j \theta\, f_j)\right]\right| \bigcup f_{SA0}\right] \theta\ f_{SA1}$$

Figure 7. Fault List Algorithms

When the inputs are normally all ones and gate G is fault-free, then F is sensitive to all inputs, and F equals the union of all input fault lists. However, if any one-valued input to G has failed open, then the corresponding input fault list can have no effect. The term $(F_i \theta f_i)$ means the fault list associated with i^{th} input is excluded when f_i is SA1 (i.e., the i^{th} input is an open circuit). The term $\bigcup_1^1 (F_i \theta f_i)$ designates the union of all fault lists on inputs that are not failed open and are normally one-valued. The gate output is normally zero so f_{SA1} is joined to the output fault list. However, if f_{SA0} exists, then all faults are excluded, and the output fault list will be empty ($F = \phi$).

For the second equation of Figure 7, first assume that all inputs are normally zero and the output is one. If there are at least two inputs ($M > 1$), then a single input open (f_i = SA1) could not affect the output because the other zero-valued input would hold the output at one. The case is the same for any upstream SA1 fault that does not affect all inputs to the gate simultaneously. If an upstream fault caused all the inputs to switch to ones, then that fault would be detectable because the gate output would go to zero (a reconvergent fanout phenomenon).

Thus, the second equation says that, for all-zero inputs, the only detectable upstream faults are those that switch all of the inputs that are not stuck at one (failed open).

Next, assume that both one and zero input values exist on the gate with the output equal to one. Initially consider a two-input gate. The output fault list, F, includes the zero-input fault list joined with that input SA1, but not including those faults on both input lists, except when the one-valued input is SA1. The reason for not including faults common to both zero-valued and one-valued inputs is that such faults would cause both inputs to change while leaving the output unchanged; hence, there would be no detection of those faults. These faults will be detected when the one-valued input is SA1 so that only the zero-valued input would be affected by upstream faults.

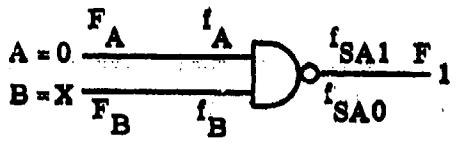
For multiple zero-valued inputs, an upstream fault will appear in F only if that fault affects all of the zero-valued inputs and none of the one-valued inputs. Hence, the SA1 faults for zero-valued inputs are joined to the corresponding upstream fault lists, then these lists for each zero-valued input are intersected. Appearance of a one-valued input is sufficient to block the list transmitted so the union of one-valued fault lists is formed except for those inputs SA1. The difference between the intersection list and union list gives the input contribution to the output fault list.

Output fault f_{SA0} is joined to F except when f_{SA1} exists, in which case the output fault list would be empty.

Don't-Know Input Effects On Fault Lists

Up to this point nothing has been said about don't-know input values in fault-list simulation. Here a don't-know nodal value is defined to be an indeterminate zero or one in contrast to the high-impedance third state of tri-state bus drivers.

Consider the four input combinations of a two-input NAND with a don't-know input.

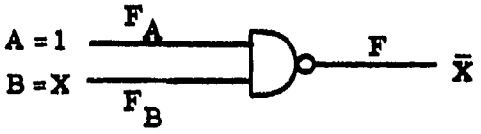


If B = 1, then $F_{A,B}$ is:

$$F_{0,1} = \left[\left|(F_A \cup f_A) - (F_B \,\theta\, f_B)\right| \cup f_{SA0}\right] \,\theta\, f_{SA1}$$

If B = 0, then

$$F_{0,0} = \left[\left|(F_A \cup f_A) \cap (F_B \cup f_B)\right| \cup f_{SA0}\right] \,\theta\, f_{SA1}$$



If B = 1, then output = 0 and

$$F_{1,1} = \left[\left|(F_A \,\theta\, f_A) \cup (F_B \,\theta\, f_B)\right| \cup f_{SA1}\right] \,\theta\, f_{SA0}$$

If B = 0, then output = 1 and

$$F_{1,0} \left[\left|(F_B \cup f_B) - (F_A \,\theta\, f_A)\right| \cup f_{SA0}\right] \,\theta\, f_{SA1}$$

The four equations for F are significantly different, which raises the question of how to represent them without specifying a value for X and for input counts greater than two. Two approaches are discussed here:

1) the minimum information approach,

2) the maximum information approach.

Minimum Information Approach

This appears to be the approach of reference 1 although no discussion was offered and probably is the approach of D4LASAR (which often sacrifices resolution for execution speed).

$$F_{0,X} = f_{SA0} \,\theta\, f_{SA1} \quad , \quad \text{output} = 1$$

$$F_{1,X} = \text{don't care} \quad , \quad \text{output} = \bar{X}$$

For $F_{0,X}$ to be correct for both values of X, it would be necessary for $\bigcap_i^0 (F_i \bigcup f_i) = \phi$. For $F_A = \phi$, either A must be driven by an external input (across whose interface no fault lists are carried) or f_{SA1} must exist. For $F_A = \phi$, F_B and f_B would have no effect. When A = 1 and B = X, then $F_{1,X}$ = don't care by definition so that don't-know outputs may propagate until either an external output is reached or a gate is reached with at least one zero-valued input. SIMUL will have marked the external output as X to be ignored. Otherwise the gate output list will be $F_{0,X}$ as defined above.

If there are more than two inputs, the same relations will hold as long as at least one input is zero-valued. If no input is a known-zero but two inputs are complementary don't-knows, then one of the two must have a zero value so the same relation will hold.

The consequence of the minimum information approach is that if there is a fault in the fault list that is computed for an external output, then it will be detected by that output. However, it may be possible in the presence of don't-knows that an external output could detect stuck-at faults, which are not in the output fault list. Thus, there would be no false detections (i.e., output error without a fault existing), but incorrect fault isolation could be possible due to incomplete listing.

Maximum Information Approach

The concept here is to specify a formula for $F_{0,X}$ and another for $F_{1,X}$ such that the computed fault list would be a least upper bound of the implied pairs of fault lists when X is specified as one or zero.

$$\text{LUB } F_{0,X} \supseteq \left[\left\{(F_A \cup f_A) - (F_B \,\theta\, f_B)\right\} \cup \left\{(F_A \cup f_A) \cap (F_B \cup f_B)\right\} \cup f_{SA0}\right] \theta\, f_{SA1}$$

For neither f_A nor f_B: $\{\ \} \cup \{\ \} = \{F_A - F_B\} \cup \{F_A \cap F_B\} = F_A$

For f_B but not f_A: $\{\ \} \cup \{\ \} = \{F_A\} \cup \{F_A \cap (F_B \cup f_B)\} = F_A \cup F_B \cup f_B$

For f_A but not f_B: $\{\ \} \cup \{\ \} = \{(F_A \cup f_A) - F_B\} \cup \{(F_A \cup f_A) \cap F_B\}$

$$= F_A \cup f_A$$

For f_A and f_B: $\{\ \} \cup \{\ \} = \{(F_A \cup f_A)\} \cup \{(F_A \cup f_A) \cap (F_B \cup f_B)\}$

$$= F_A \cup f_A \cup F_B \cup f_B$$

$$\text{LUB } F_{0,X} = \left[F_A \cup f_A \cup F_B \cup f_B \cup f_{SA0}\right] \theta\, f_{SA1}$$

For $F_{1,X}$ there is a problem of what to do about the output stuck-ats as both output values 0, 1 may be obtained depending upon the value of X. The LUB of these terms is taken to be $(f_{SA1} \cap \bar{f}_{SA0}) \cup (\bar{f}_{SA1} \cap f_{SA0})$

$$\text{LUB } F_{1,X} \supseteq \left\{(F_A \,\theta\, f_A) \cup (F_B \,\theta\, f_B)\right\} \cup \left\{(F_B \cup f_B) - (F_A \,\theta\, f_A)\right\} \cup f_{SA1} \cap f_{SA0} \cup f_{SA1} \cap f_{SA0}$$

For neither f_A nor f_B: $\{\ \} \cup \{\ \} = \{F_A \cup F_B\} \cup \{F_B - F_A\} = F_A \cup F_B$

For f_B but not f_A: $\{\ \} \cup \{\ \} = \{F_A\} \cup \{(F_B \cup f_B) - F_A\} = F_A \cup F_B \cup f_B$

For f_A but not f_B: $\{\ \} \cup \{\ \} = \{F_B\} \cup \{F_B\} = F_B$

For f_A and f_B: $\{\ \} \cup \{\ \} = \{\phi\} \cup \{F_B \cup f_B\} = F_B \cup f_B$

$$\text{LUB } F_{1,X} = F_A \cup F_B \cup f_B \cup (f_{SA1} \cap \bar{f}_{SA0}) \cup (\bar{f}_{SA1} \cap f_{SA0})$$

The least upper bounds assure that every fault that can be detected by an external output will appear in the output fault list in the presence of don't knows. However, some of the listed faults will not be detected by all of the possible input combinations. The consequence of this is an inclusion of ambiguities in the output fault lists so that a dictionary derived from them would provide less isolation resolution than would be possible if the don't-know values were known.

VIII. REDUCE

The module REDUCE determines the extent to which faults may be differentiated by the specified test set. REDUCE also generates the X, Y, Z fault-isolation tables. The differentiation of detected faults is accomplished by intersecting DYSOGN fault lists and their complements.

As an illustration, assume a network with two outputs is exercised with two tests. Let A, B, C, D be the fault lists respectively for first test and first output, first test and second output, second test and first output, second test and second output. For any fault to be suspect, it must be in all of the fault lists associated with every output error.

The outputs may produce errors in any combination depending on the existing fault and the test that is executed. For two outputs, there are three combinations of error patterns: $E_1 \cap \bar{E}_2$, $\bar{E}_1 \cap E_2$, $E_1 \cap E_2$. The maximal fault isolation obtainable from each test is given by the following table.

Test No.	Error Pattern	Implied Faults
1	$E_1 \cap \bar{E}_2$	$A \cap \bar{B}$
1	$\bar{E}_1 \cap E_2$	$\bar{A} \cap B$
1	$E_1 \cap E_2$	$A \cap B$
2	$E_1 \cap \bar{E}_2$	$C \cap \bar{D}$
2	$\bar{E}_1 \cap E_2$	$\bar{C} \cap D$
2	$E_1 \cap E_2$	$C \cap D$

The maximum isolation information obtainable from the whole test set is obtained by first observing the sequence of output error patterns obtained when the test set is executed, then intersecting the fault sets associated with the observed error patterns. The following table is derived from the foregoing table.

Error Patterns		Implied
First	Second	Fault Set
$E_1 \cap \bar{E}_2$	$E_1 \cap \bar{E}_2$	$A \cap \bar{B} \cap C \cap \bar{D}$
$E_1 \cap \bar{E}_2$	$\bar{E}_1 \cap E_2$	$A \cap \bar{B} \cap \bar{C} \cap D$
$E_1 \cap \bar{E}_2$	$E_1 \cap E_2$	$A \cap \bar{B} \cap C \cap D$
$\bar{E}_1 \cap E_2$	$E_1 \cap \bar{E}_2$	$\bar{A} \cap B \cap C \cap \bar{D}$
$\bar{E}_1 \cap E_2$	$\bar{E}_1 \cap E_2$	$\bar{A} \cap B \cap \bar{C} \cap D$
$\bar{E}_1 \cap E_2$	$E_1 \cap E_2$	$\bar{A} \cap B \cap C \cap D$
$E_1 \cap E_2$	$E_1 \cap \bar{E}_2$	$A \cap B \cap C \cap \bar{D}$
$E_1 \cap E_2$	$\bar{E}_1 \cap E_2$	$A \cap B \cap \bar{C} \cap D$
$E_1 \cap E_2$	$E_1 \cap E_2$	$A \cap B \cap C \cap D$

If an error is detected but the implied fault set is empty, then either 1) the causative fault was not simulated (possibly due to don't knows) or 2) the fault causes inconsistent errors (definitely not stuck), or 3) the simulation results are incorrect due to modelling or programming error or host-computer error when executing DYSOGN.

Where outputs are don't-know Xs, they are ignored. This is equivalent to intersecting the set of all faults.

The upper bound on the number of uniquely isolated fault sets is $N_T \cdot (-1 + 2 \exp N_E)$, where N_T is the number of tests and N_E is the number of observable outputs.

The algorithm actually used by D4LASAR's REDUCE has not been disclosed, but it is known that the SMC-3100 host computer is able to execute AND operations (which correspond to intersection) on vectors of exceptional length.

IX. CONCLUDING COMMENTS

This report is intended as an introduction to the D4LASAR test-generation system, with emphasis on a mechanistic background. The report endeavors to show how D4LASAR works, some things D4LASAR does well or poorly, and additional features that would be desirable.

The serious user will need to learn the programming details, preferrably via a formal course and then through continued usage. The default values of options in D4LASAR have been chosen with sufficient care that much useful test generation can be accomplished by a person with a shallow understanding of the process. However, things do not always go well, in which case there is no substitute for understanding.

There is a great deal to learn about modelling problems, and this report does not touch on this aspect. Understanding of work-arounds is important. Test-generation programs use digital computers with their binary processes. However, many circuits, e.g., edge-triggered flip-flops and transmission gates, incorporate analog processes that must be modelled and, in some way, adapted to the computer's limitations. STIMGN models may require work-arounds that are not required by SIMUL and DYSOGN because of the difference in time representation. Tri-state bus drivers require awkward work-arounds because the program is not written to accommodate two-way signal flow on a single wire. (This writer has heard of no simulator program with that capability). If it is desired to simulate a well-designed synchronous machine, a considerable amount of manual labor is required with D4LASAR to generate the equivalent of a clock. If a large network is partitioned, to permit application of D4LASAR to manageable pieces, then there may be considerable difficulty in maintaining the timing relations at the partition interfaces. These remarks are made to emphasize the need for study to become proficient in the application of D4LASAR (or other such programs) to wide ranges of real test problems.

Two more warning comments are in order. The Complementary Metal Oxide on Silicon (CMOS) technology applications are growing rapidly due to the advantageous speed-power product. These gates have "complementary" logic, which is really dual logic (AND and OR are duals), at their outputs. D4LASAR and most (perhaps all) of its competition are designed for TTL gates and do not accurately represent CMOS gates. It seems that the fault coverage for CMOS will be less than its TTL equivalent, but we do not seem to know what to do about it. Further study is needed for modelling CMOS devices.

D4LASAR depends on gate models. According to information obtained from Digitest Corp in March, 1978, the gate limits, as functions of the number of words of core in the host SMC-3100 computer, are as follows:

Gate Limits

Module	Words of Core	
	196,608	524,244
STIMGN	5,800	20,700
SIMUL	7 750	28,000
DYSOGN	4,500	14,800

Some large-scale integrated (LSI) devices, with more than 5,000 NAND-equivalent gates, already exist. The number of gates on a board may be many tens of thousands. Hence, some single chips already overwhelm the minimal D4LASAR system, and there are boards that overwhelm the maximal D4LASAR system. The run times can require days when the chip or board barely fits into the system limits. D4LASAR is a finely tuned program so there is little hope for large improvements in run time through further tuning. A faster host computer would help the run time, but the cost would not diminish in proportion to the speed increase. Increased use of bulk storage would allow much larger gate counts but with significant run time increases.

Such considerations show that D4LASAR and its competition are close to being obsolete because the growth of LSI devices continues much faster than the growth of the test-program capability. The problem stems primarily from gate-level modelling; there is too much information to be processed.

It will become necessary to give up gate models except for use by LSI chip designers. Reference 7 offers alternative approaches. Loss of the fine structure of gate models will severely limit our ability to assure high fault coverage and to account for timing problems. Perhaps the ultimate solution is to provide self-testing devices using hardware redundancy techniques. As we approach that goal, there will surely be many attempts to find compromises between gate-modelled software and fault-tolerant hardware.

REFERENCES

1. Armstrong, D.B.,"On finding a nearly minimal set of fault detection tests for combinational logic nets," IEEE Trans On Electr Comp, Vol EC-15, No. 1, Feb 1966, pp 66-73.

2. Mealy, G.H., "A Method for Synthesizing Sequential Circuits," BSTJ Vol. 34, Sept. 1955, pp 1045-1079.

3. Mott, T.H. Jr., "Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of the Prime Implicants," IRE Trans On Electr Comp Vol EC-9, No. 2, June 1960, pp 245-252.

4. Chappell, Elmandorf, and Schmidt, "Logic Circuit Simulators," BSTJ Vol 53, No. 8, Oct. 1974, pg 1463.

5. Eichelberger, E.B., "Hazard Detection in Combinational and Sequential Switching Circuits," IBMJ, March 1965.

6. "F-16 Automatic Test Program Generator Evaluation," Vol 1, Report 16PR462, Gen Dynamics Corp., A.F. Contract F33657-75-C-0310, 28 Nov., 1977.

APPENDIX A: EXAMPLE

This example is strictly combinatorial and was run on D4LASAR but not under ALEC to emphasize the procedures of the main modules of D4LASAR.

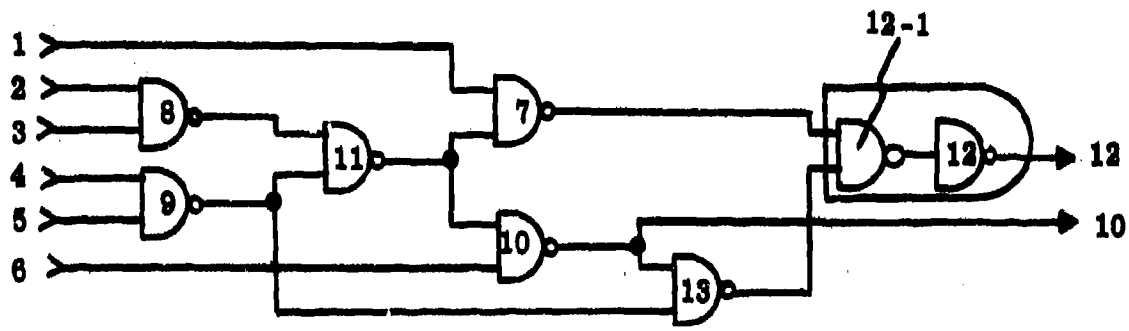


Figure 8. Logical Model

Model Deck

```
NAME = EXAMPLE
MODEL/
7NA/1,11//, 8NA/2,3//, 9NA/4,5//, 10NA/11,6//,
11NA/8,9//, 12AN/7,13//, 13NA/10,9//
INPUT/1,2,3,4,5,6/
OUTPUT/10,12/
```

Component Ordering For Minimum Feedback (by INPUT)

```
1      9NA
1      8NA
2     11NA
3     10NA
3      7NA
4     12AN
4     13NA
```

As gate 12 output is a function of gate 13 output, 12 should be last.

STIMGN Tests

Test No.	Input Values 1	2	3	4	5	6	Source Output	Output 10	12
1	X	0	X	1	1	1	10	0	(X)
2	X	1	1	0	X	1	10	0	(X)
3	X	X	X	1	1	0	10	1	(X)
4	X	0	1	0	1	1	10	1	(1)
5	X	1	0	0	1	1	10	1	(1)
6	X	0	X	1	0	1	10	1	(X)
7	1	X	X	1	1	X	12	(X)	0
8	0	X	X	0	X	0	12	(1)	0
9	0	X	X	1	1	0	12	(1)	1
10	0	1	1	0	X	1	12	(0)	1

Bracketed outputs are not given by STIMGN and were hand calculated.

STIMGN Detected Faults

Test No.	New Detects Per STIMGN
1	10 SAI, 6SAO, 11 SAO, 9*11, 9 SA1, 4 SAO, 5 SAO
2	8*11 , 8SA1, 2 SAO, 3 SAO
3	10 SAO, 6*10, 6 SA1
4	11*10, 11 SA1, 9 SAO, 8 SAO, 4*9, 4 SA1, 2*108, 2 SA1
5	3*8, 3 SA1
6	5*9, 5 SA1
7	12 SA1, 12-1 SAO, 7*12-1, 7 SA1, 1 SAO
8	13*12-1, 13 SA1
9	12 SAO, 12-1*12. 12-1 SA1, 7 SAO, 13 SAO, 1*7, 1SA1, 9*13
10	10*13

Undetected (and undetectable) 11*7

8*11 designates open line from 8 to 11 (equivalent to SA1 input).
12-1 is first of two NANDS composing the AND labelled 12.

Comments On STIMGN Example

Output 10 is selected as the first source output because it is listed first on the OUTPUT control card.

The order of input assignments, e.g. critical vs necessary, on each gate is determined by the INPUT module in its routine for minimizing feedback. This example has no feedback but is reordered anyway. Why INPUT did not list gate 12 as a fifth level is not understood by this writer.

The first value assigned to any source output is zero. Thus, for the first test, gate-10 output is zero which requires a (critical) all-ones input. External input 6 and gate-11 outputs must be ones, so gate 11 must have a single-zero input. Gate 9 is ordered ahead of gate 8, so gate-9 output is back-driven to a critical zero and gate-8 output to a necessary one. Then external inputs 4 and 5 must both be (critical) ones. External input 2 becomes a necessary zero while input 3 remains "X" (designating a don't care in this case). Incidentally, gate 10 forces a zero input to gate 13, and gate 11 forces a one input to gate 7, but these fanouts do not reconverge so there is no possibility for a logical contradiction.

A minor question of STIMGN procedure arises. Mr. Bruce Pomeroy of Digitest Corp. has stated that STIMGN does not pursue parallel critical paths, i.e., it does not consider all upstream branchings from gates with all-one input, but that STIMGN does have some look-ahead capability. In the example of test one above, STIMGN does find all of the possible detectable faults including those of gates 8, 9, 11. Without the undefined look-ahead feature, STIMGN would designate external input 6 as a critical 1 and gate-11 output as a necessary one. In such an analysis, the external test vector would be XXX111 (don't-care gate 8) and none of the faults of gates 8, 9, or 11 would be detected by test one. This exemplary model is too simple to determine the number of levels of lookahead.

Tests which do not result in new detects will not be listed. This is demonstrated by detailed analysis of test 9 and 10. Test 9 begins with source 12 output changed to one. Then gates 7, 10, and 13 would be toggled in turn. However, toggling 7 and 10 produces no new detect and, therefore, no test is listed. Test 10 derives from toggling gate 13.

The STIMGN control ordering, which resulted in the specified sequence of of ten tests are: 1) source 10 set to zero, 2) toggle gate 11, 3) source 10 set to 1, 4) toggle gate 10, 5) toggle gate 8, 6) toggle gate 9, 7) source 12 set to zero, 8) toggle gate 12, 9) source 12 set to one, 10) toggle gate 13.

STIMGN execution time was eight seconds.

OVRLAY Patterns

STIMGN TESTS	OVRLAY TEST NO.	WITH DON'T CARES	ACTUAL PATTERNS
1, 7	1	1 0 X 1 1 1	1 0 1 1 1 1
2, 10	2	0 1 1 0 X 1	0 1 1 0 1 1
3, 9	3	0 X X 1 1 0	0 0 1 1 1 0
4	4	X 0 1 0 1 1	0 0 1 0 1 1
5	5	X 1 0 0 1 1	0 1 0 0 1 1
6	6	X 0 X 1 0 1	0 0 0 1 0 1
8	7	0 X X 0 X 0	0 0 0 0 0 0

All don't cares remaining after overlaying are assigned so as to minimize the number of changes. The X of test 1 is set to the one of test 2. The X of test 2 is set to the one of test 3. As it happens, the first X of test 3 is the only one preceded by one and followed by zero, and it is set to the latter. The Xs of test 8 are set to the preceding values.

OVRLAY execution time was four seconds.

SIMUL

There is no latch in the example, hence there can be no race nor hazard. Thus, no buffer pattern was added, and no node has a don't-know value for any test. Unlike STIMGN, SIMUL calculates all (two) output values for each test.

SIMUL execution time was seven seconds.

DYSOGN

Test 1 (OVRLAY: 101111) will be used as an example of fault-list simulation. Remember that gate ordering is 9, 8, 11, 10, 7, 13, 12. Corresponding gate output values are 0, 1, 1, 0, 0, 1, 0. The following fault lists were manually calculated and are believed to be the same as DYSOGN's output. No DYSOGN option is available to permit checking this.

DYSOGN execution time was five seconds.

Fault Lists

$$F9 = F4 \cup F5 \cup 9_{SA1}$$

$$F8 = F2 \cup (2*8) \cup 8_{SA0}$$

$$F11 = \left\{ [F9 \cup (9*11)] \; \theta \; F8 \right\} \cup 11_{SA0}$$

$$= F4 \cup F5 \cup 9_{SA1} \cup (9*11) \cup 11_{SA0}$$

$$F10 = F6 \cup F11 \cup 10_{SA1}$$

$$= F4 \cup F5 \cup F6 \cup 9_{SA1} \cup (9*11) \cup 11_{SA0} \cup 10_{SA1}$$

$$F7 = F1 \cup F11 \cup 7_{SA1}$$

$$= F1 \cup F4 \cup F5 \cup 9_{SA1} \cup (9*11) \cup 11_{SA0} \cup 7_{SA1}$$

$$F13 = \left\{ [F9 \cup (9*13)] \cap [F10 \cup (10*13)] \right\} \cup 13_{SA0}$$

$$= F4 \cup F5 \cup 9_{SA1} \cup 13_{SA0}$$

$$F(12\text{-}1) = \left\{ [F7 \cup (7*12\text{-}1)] \; \theta \; F_{13} \right\} \cup (12\text{-}1)_{SA0}$$

$$= F1 \cup (9*11) \cup 11_{SA0} \cup 7_{SA} \cup (7*12\text{-}1) \cup (12\text{-}1)_{SA0}$$

$$F12 = F(12\text{-}1) \cup 12_{SA1}$$

The fault list detected by test 1 under single-fault assumption is that provided by DYSOGN:

$$F10 \cup F12 = F1 \cup F4 \cup F5 \cup F6 \cup 9_{SA1} \cup (9*11) \cup 11_{SA0} \cup 7_{SA1} \cup (7*12\text{-}1) \cup 10_{SA1} \cup (12\text{-}1)_{SA0} \cup 12_{SA1}$$

TABLE OF DETECTED FAULTS VS TEST NO.

FAULT \ TEST NO.	1	2	3	4	5	6	7
F1	1	1	1				
F4	1		1	1	1		
F5	1		1			1	
F6	1	1	1				
9_{SA1}	1		1				
(9*11)	1						
11_{SA0}	1	1					
7_{SA1}	1						
(7*12-1)	1						
10_{SA1}	1						
$(12-1)_{SA0}$	1						
12_{SA1}	1						
F2		1		1			
F3		1		1			
1*7		1	1				
8_{SA1}		1					
8*11		1					
7_{SA0}		1	1				
10*13		1					
13_{SA0}		1	1				
$(12-1)_{SA1}$		1	1				
(12-1)*12		1	1				
12_{SA0}		1	1				
6*10			1				
9*13			1				
10_{SA0}			1	1	1	1	1
4*9				1	1		
2*8				1			
9_{SA0}				1	1	1	1
8_{SA0}				1	1	1	
11_{SA1}				1	1	1	
11*10				1	1	1	
13_{SA1}				1	1	1	1
13*12-1				1	1	1	1
$(12-1)_{SA0}$				1	1	1	1
12_{SA1}				1	1	1	1
3*8					1		
5*9						1	

11*7	Not detected
Test 7	Not needed

APPENDIX B. CRITICAL PATHS WITH RECONVERGENT FANOUT

If there were no reconvergent fanout, STIMGN's back-driven critical paths would have a much easier task. Reconvergent fanout allows one node to drive another through two or more parallel intermediate paths. When discussing pulse generation, the delays of these paths are important. However, here delays are of no concern, but the parity (odd or even gate count) is important.

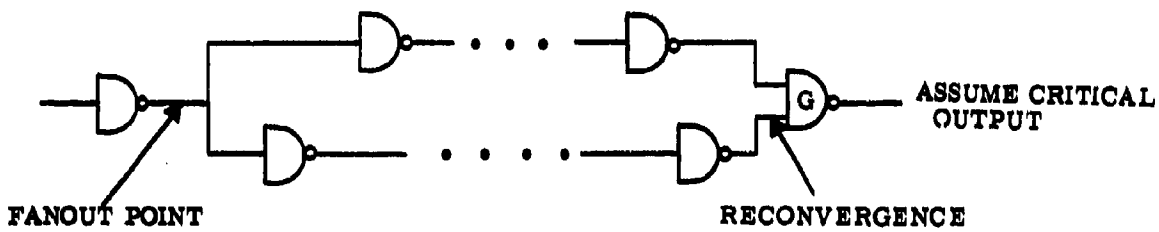


Figure 9. Reconvergent Fanout

If the parities of both branches are different, then the inputs to G will be complementary. The zero-valued path will be "critical" while the other path is "necessary". That is, errors in the critical path will be detected but not in the necessary path. Upstream of the fanout point there can be no criticality through gate G because the effect of such upstream faults would change both complementary inputs to G leaving its output unchanged.

If the parities of both branches are the same, then the inputs to G are either both one or both zero. If both are one, all of the gates are critical, which is not remarkable. However, if both inputs to G are zero, then no fault in the parallel paths can be detected because the other unfaulted path would hold the G output value at one. All gates upstream of the fanout point are critical because they would cause both of the inputs to G to change from 0, 0 to 1, 1 with a detectable change in the output of G.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.