A CMOS LSSD test generation system

by D. Leet
P. Shearon
R. France

Automatic test pattern generators based on the stuck-fault concept are theoretically inadequate in their ability to generate test patterns for CMOS circuits. A new set of pin faults, called CMOS faults, is discussed that can represent the necessary test pattern sequences for these circuits. Processing of these faults by a new test pattern generator, called the Enhanced Test Generator (ETG), is also described.

Introduction

The test generation problem can be stated as follows.

Given A system, hereafter called a *design*, consisting of *blocks* interconnected by *nets*.

The blocks are of three types:

1. PI blocks—Single output, directly controllable.
2. PO blocks—Single input, directly observable.
3. Logic blocks—n input nodes, $n > 0$, m output nodes, $m > 0$. The ith block has K_{i+1} transfer functions, all of which are Boolean functions of limited time domain. One of these functions is called the *good machine*, or GM, function, and is intended to be the transfer function of the block without any faults in it. The other K_i functions are called *failing machine*, or FM, functions, and are associated with fault classes of the block. Some transfer functions can have a finite amount of delay, but no feedback. Others can have feedback and store information.

The nets are almost always assumed to be unidirectional, with no logical function and zero delay.

Problem Find an algorithm that, for each defined FM function, generates at least one *design* behavior that is different from that of the GM.

Algorithms that attempt to solve this problem are usually evaluated according to a measure that reduces to the percentage of FMs that are detected, assuming that only one FM can occur at a time. Other performance criteria used to evaluate test pattern generation algorithms include

- The time required to generate the patterns.
- The resources (computer time, computer memory, computer DASD, number of people, and the people skills) required to generate the patterns.
- The time required to apply the test patterns and determine that the design is entirely GM.
- The average time to apply the test patterns and determine that the design is entirely GM, assuming a mix of both GM and FM designs.
- The probability that a design classified as GM cannot function in its assigned environment at a later time. (This criterion is, to a large extent, dependent on the adequacy of the FM functions to incorporate the significant failure modes of the design.)

This paper makes the following assumptions about a design:

- The failure modes considered are only those that can lead

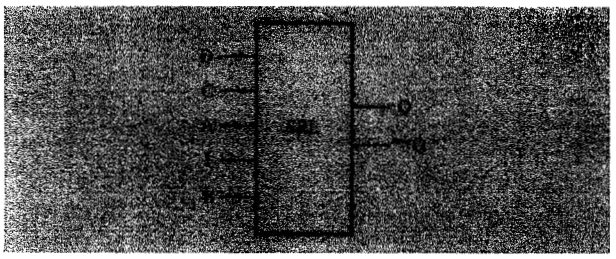

Figure 1

SRL block.

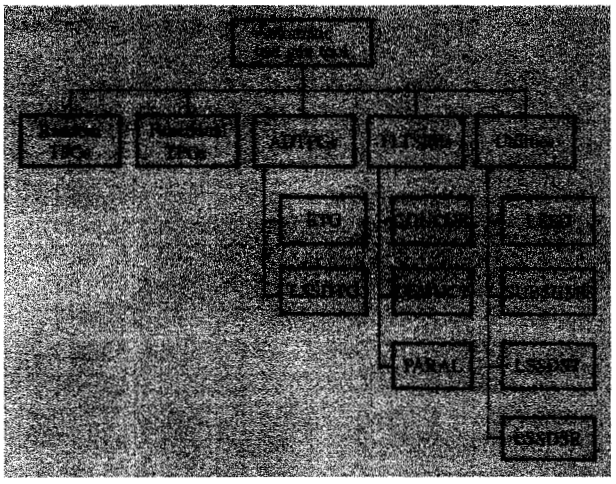

Figure 2

High-Level Function of the Automatic Test System.

to a steady state difference between the GM and FMs (static-fault assumption).

- Only one defective block at a time (single-fault assumption).
- Nets between blocks do not short except at block input nodes.
- The stimulus and response values are taken from the set (0, 1, *X*, *H*). 0 and 1 are the standard logic values. *X* means an unknown logic value. *H* means a high-impedance state.
- The logic blocks are A, AI, O, OI, DOTA, DOTO, DOTT, and SRL. The first four blocks are the standard logic primitives. The DOTA and DOTO blocks have transfer functions that are the same as the A and O blocks, respectively, but they can be different from the primitive functions in some other respects, such as their fault model. The DOTT block is the only block that can process and generate an *H* value.

The SRL is an LSSD-compatible latch, such as the polarity hold latch, whose block diagram is shown in **Figure 1.** It is a level-sensitive master slave latch with a two-port first stage, called L1, feeding a second stage called L2. There are two clocks driving L1, C and A, with the C clock gating D and the A clock gating I. There is one clock gating L2, the B clock, which does not overlap in time either A or B. Operationally, C is the system clock and D is the data port. A is called the scan clock.

The SRLs are connected into one or more shift registers by chaining the I inputs. A design input and output are designated to feed the first SRL and receive the last SRL output, respectively, for each shift register. This arrangement offers a way to directly observe and control every state in the design.

- The design obeys the LSSD design rules. These rules ensure that the various clocks operate properly and can be tested and that scanning occurs with complete controllability and observability. (For more information on the LSSD design methodology, see [1] and [2].)

Most logic chip manufacturers have a collection of hardware and software tools that help them generate test patterns [3, 4]. Some have integrated their software tools into a large software system similar to the one discussed in this paper. In fact, an executive program might be used to control and select the appropriate tool for a given situation [5].

In **Figure 2,** the tools are divided into five classes, three of which are associated with test pattern generation (TPG): random (random TPG), functional behavior (functional TPG), and automatic deterministic (ADTPG). The other two classes are fault simulators and utilities.

The subject of this paper is a new ADTPG called the Enhanced Test Generator, or ETG. ETG is like most other ADTPGs in that it applies defined objectives (stimulus sequence and GM and FM responses) to a block, or BLOCK, in capital letters, and then attempts to force BLOCK's response to at least one point of observability in the design and BLOCK's stimulus sequence to points of controllability in the design.

Like other ADTPGs, ETG can generate test patterns for objectives represented as stuck faults. ETG, however, has a unique capability: It can automatically generate test patterns for the special faults found in CMOS circuits called CMOS open faults. Since the stuck-fault concept is in general use, ETG's operation with respect to stuck faults is discussed first. Then the concept of CMOS pin faults is introduced and the additional capability available in ETG to process these faults is described.

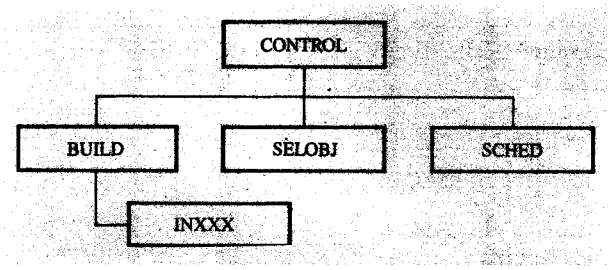

Figure 3

High-Level Function of an ADTPG.

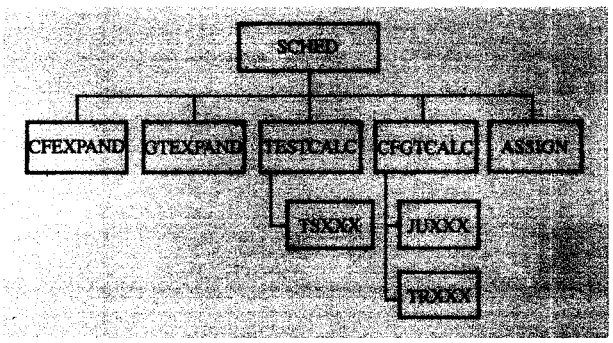

Figure 4

High-Level Function of the Scheduler.

The organization of ETG

In discussing ETG, the following data structures are assumed:

LMOD	defines the blocks, nets and connectivity of the design.
BUGLIST	contains the following information for each objective: • Design objective number. • Code defining the objective. • Definition of where the objective is located in the design.
NETIMAGE	is a structure of lists for every net in the design that consists of elements containing the GM and FM values on the net and the time image in which they were assigned. Miscellaneous flags are also contained in each element.
BLKLIST	is a list structure of blocks to be processed.

ETG is organized as shown in **Figures 3** and **4**. The functions in these figures are as follows:

BUILD	builds and initializes the internal data structures.
SELOBJ	selects the objectives to be processed.
SCHED	processes the objectives. Decisions are remade whenever a conflict occurs, discarding data that are judged unnecessary.
TSXXX	applies the stimulus and response values (GM and FM) to BLOCK.
TESTCALC	invokes a TSXXX based on BOOK's function.
CFEXPAND	adds blocks that feed BLOCK to BLKLIST.
GTEXPAND	adds blocks that are fed by BLOCK to BLKLIST.
CFGTCALC	selects other blocks to transmit or justify values through.
TRXXX	transmits values through blocks.
JUXXX	justifies values through blocks. (It finds a stimulus that produces the given response.)
ASSIGN	manages the internal data structures that contain the design's stimulus and response values for this objective.

The XXX in TSXXX, TRXXX, and JUXXX indicates that there are several of these functions in ETG for the various blocks defined in the design.

Stuck-fault test pattern generation

• *Implementation of the stuck-fault concept in TSXXX*

The ADTPGs generally available today rely on the stuck faults on the pins of primitive blocks to represent objectives [6–8]. Two reasons for this are

- Historically, single-stuck-fault models of designs have resulted in good-quality product.
- The stuck-fault concept leads to an efficient implementation in an ADTPG.

The last point is elaborated first.

The stuck-fault approach makes a basic assumption about the FM transfer functions: They can be characterized as being the GM transfer function except that a single input or output is stuck at a logic value. For example, the set of unique FM transfer functions for a three-input AI gate, with inputs A, B, and C and output Z, is summarized in **Table 1**. The "–" in the Stimulus columns mean that a 0 can occur on one or more of the inputs. In all columns, the GM value is on the left of the "/" and the FM value is on the right.

Table 1 AI test patterns.

Stuck fault	*Stimulus*			*Response Z, G/F*
	A	*B*	*C*	
Asa1	0/	/1	/1	1/0
Bsa1	/1	0/	/1	1/0
Csa1	/1	/1	0/	1/0
Zsa0	-/	-/	-/	1/0
Zsa1	1/	1/	1/	0/1

Table 2 AI test patterns.

Stuck fault	*Stimulus*		*Response O, G/F*
	I	$\neg I$	
Is1	0/	/1	1/0
Os0	-/	-/	1/0
Os1	1/	1/	0/1

Note how compactly the FM transfer functions can be defined as the difference between the GM and FM transfer functions. Also, note that both the GM and the FM are defined in both stimulus and response columns. This is not normally done in the literature, but it emphasizes the essential differences between the GM and FMs, as well as resolving conflicts in circuits with feedback.

Table 1 suggests an efficient combination of database format and program coding for applying patterns to an AI block. The database specifying the faults need contain only the stuck-fault name, the pin the stuck fault is on, and the block the pin is on. A TSXXX function that might be written for the AI primitive could take this information and implement the table as a case statement on the stuck fault.

Table 1 implies that a separate copy of the function is not needed for each size AI block in the design. In all cases, the decision table says to do something to one input, and to do something else to every other pin. As a matter of fact, **Table 2** is a shorter way to write Table 1 for all AI sizes.

A table can be constructed for all the primitive blocks, not just the AI block, by using the concept of controlling value. A controlling value is a value on a single input that forces the output response to be a certain value, regardless of the other input values. The controlling value for A and AI blocks is 0; the controlling value for O and OI blocks is 1.

With CV representing controlling value and NV representing noncontrolling value, **Table 3** is the combined decision table for all primitives. The response column in this table defines the outputs for the true primitives A and O. The inverted primitives AI and OI are assigned values in which CV and NV are reversed.

Let TSAOI be the TSXXX function associated with the primitive blocks. Its task is to assign values, based on this table, to nets feeding a block. (The values are placed in the NETIMAGE structure.)

The following points should be considered when implementing TSAOI:

- A blank in Table 3 means that nothing is assigned to that net and no checks are made.
- If an assignment is required, the value already on the net cannot conflict with the value to be assigned. If it does, this should be communicated to higher levels of the program.
- The OSNC entry should be interpreted as follows: Apply a CV to one or more block inputs. A decision must be made on which one. A mechanism should exist to keep track of decisions, so that if one decision leads to a conflict somewhere in the system, other decisions, if they exist, can be tried.

- *Transmitting responses to POs or SRLs*

Once the test values have been assigned to BLOCK's nets, it is the responsibility of the ADTPG to transmit the response values to the points of observability, which are SRL or PO blocks when the design obeys the LSSD design rules. In doing this, the ADTPG can be expected to call TRXXX functions repeatedly to transmit test values through blocks.

These are some important points that were considered when the TRXXX functions were implemented in ETG:

1. One of the flags, TFLAG, in the NETIMAGE element can indicate that the value on the net is a test value that should be transmitted through the block.
2. Any primitive block that has a net feeding it with a TFLAG should apply noncontrolling values to all other inputs in order to pass the value through the block. **Table 4** is the decision table for this assignment for all primitive blocks. The response column contains the values for the noninverted primitives A and O. The values for the inverted primitives have NC in place of CV and vice versa.

- *Justifying stimuli back to PIs or SRLs*

The stimuli to BLOCK, plus all stimuli assigned during the transmit operation, must eventually be justified back to points of controllability, which are PI or SRL blocks in LSSD designs.

The JUXXX functions are responsible for this operation for a given block. **Table 5** is the decision table implemented in the JUAOI function associated with the primitive blocks. The table should be interpreted as follows:

1. Assignments should not be attempted where entries are blank.

2. If the stimulus value is NC, that value should be applied to all input pins.
3. If the stimulus value is CV, that value should be applied to one or more input pins. A decision is required and a mechanism must exist for tracking these decisions and remaking them when a conflict arises in the system.

• *Construction and validation of LSSD test patterns*

The discussion in the previous paragraphs assumes that the design obeys the LSSD design rules. Thus the test pattern generator can assume that the latches are PIs and/or POs.

After the patterns are generated, one for each objective, they must be formatted into patterns consistent with the design's LSSD structure, complete with all system and scan clock stimuli, and then passed through a simulator to verify that they are valid patterns. Conceptually, the following procedure is used by ETG to accomplish this:

1. The utility LSSDST (see Fig. 2) is called to define the stimuli necessary to set the design up for scanning. After the stimuli are applied, the design is said to be in its "stability state."
2. The utility LSSDSR is called to define a set of test patterns for the shift registers themselves. These patterns may or may not be simulated.
3. The test pattern generator is called, perhaps several times either for different strategies or just to break the patterns up into smaller groups. Each time the test pattern generator is called:
   a. The test pattern generator generates tests for the objectives in the list passed to it.
   b. The patterns are subsumed where possible using the utility SUBSUME. (This reduces the test pattern count, but it means that there can be multiple diagnostics for a given test pattern.)
   c. LSSD tester loops are constructed using the utility LSSD. (A tester loop is defined below.)
   d. The tester loops are sent to the fault simulator. This simulator might use a concurrent, deductive, or parallel approach to fault simulation (CONCUR, DEDUCT, and PARAL in Fig. 2. See [9].). The fault simulator applies the defined stimuli to the design and marks off a fault whenever it is detected. It defines the responses of the design to the stimuli. Each marked-off fault results in a diagnostic being recorded in a diagnostic data set. (It should be noted that the fault simulator will mark off any faults that are detected by a test pattern. This means that even though the first tester loop may have been generated for one objective, it might in fact be effective in detecting several other faults. This in turn means that the subsequent patterns that were generated for those faults are no longer required, unless, of course, they too detect faults other than the one for which they were generated.)

Table 3 Test patterns for all primitives.

Stuck fault	*Stimulus*		*Response T*
	I	$\neg I$	
IsNC	CV/	/NC	CV/NC
OsCV	NC/	NC/	NC/CV
OsNC	CV/	CV/	CV/NC

Table 4 TRXXX decision table for all primitives.

Value to be transmitted	*Value on other pins*	*Response*
CV/NC	/NC	CV/NC
NC/CV	NC/	NC/CV

Table 5 JUXXX decision table for all primitives.

Defined response	*Required stimuli*
NC/	NC/
CV/	CV/
/NC	/NC
/CV	/CV

4. Tester loops that have no diagnostics are discarded.

A typical tester loop has the following format:

1. Load the SRLs.
   a. Apply the stimuli necessary to establish the stability state.
   b. Order the values assigned to the SRLs according to their location in the shift registers.
   c. Provide a series of A and B clocks and apply stimuli to scan-in ports to scan values into the SRLs. (The values have been previously assigned to the SRLs according to their order in the SRL scan chain.)
2. Apply values to the PIs.
3. Provide a CB clock pair to store the response of the design to the stimulus.
4. Measure the PO responses.
5. Unload the SRLs.
   a. Apply the stimuli necessary to establish the stability state.
   b. Provide a series of A and B clocks to scan values out of the SRLs, measuring the scan-out ports after each pair of clocks.

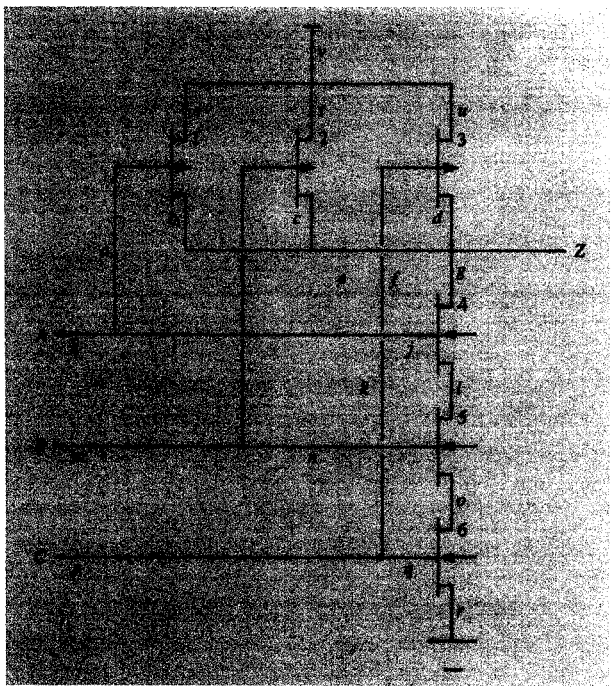

Figure 5

Three-input standard CMOS NAND gate.

CMOS test pattern generation

• *CMOS pin faults, the TSXXX routine, and the BUGLIST*
Although the CMOS technology has many design and technology advantages over the common bipolar and nMOS technologies [10], it has a potentially serious testing disadvantage, particularly in low-yield situations: the CMOS open fault [11, 12].

Figure 5 is a standard CMOS three-input AI gate. It is "standard" in the sense that the circuit has pairs of p- and n-type transistors, each pair driven by the same input. In this figure, all the transistors have been numbered and all the wire segments have been lettered.

This circuit and the circuits that implement the other primitives have been analyzed for their behavior, given these device-level faults in the circuit.

- Open transistors.
- Short transistors.
- Open wire segments.
- Two wire segments shorted together.

To summarize the results for the given AI gate: The faults could be classified into the following three major groups based on the stimuli necessary to detect them (the number in parentheses is the percentage of all the faults that belong to this group):

1. Stuck-at faults (24 percent) exhibit stuck-fault behavior.
2. CMOS open faults (47 percent) exhibit CMOS open-fault behavior. That is, in order to guarantee a test for the fault, the output of the circuit must be either charged or discharged to the FM value and then a second stimulus must be applied in an attempt to change this value.
3. Shorted device faults (27 percent) exhibit behavior that can be modeled as one of the devices shorted from source to drain. These faults may or may not be dc-detectable, depending on technology specifics.

The CMOS open fault is important for two reasons. First, with the right sequence of stimuli, it is dc-testable. As a matter of fact, if the ADTPG generated the appropriate clocks, it is likely that at least some of these faults can be serendipitously caught. Second, without any information on defect densities, it must be assumed that this fault constituted a significant fraction of the faults in the design.

This was the motivation for the development of ETG, which actively generates test patterns for CMOS open faults, and the associated fault simulators, which grade the resulting patterns for effectiveness.

Key to the development of ETG was the realization that CMOS open faults can be represented in basically the same way as stuck faults; that is, as pin faults on a primitive block. This realization is based on the observation that the open device faults can be further classified into one of three groups:

1. Inclusive input faults—those that require a transition on the specified input pin. An example of this fault in Fig. 5 is device 1 open from source to drain. To test this fault, the output must first be discharged with a stimulus of (A, B, C) = (1, 1, 1), and then device 1 must be used to charge the output with a stimulus of (0, 1, 1). Note that devices 2 and 3 are held off to ensure that it is device 1 that does the charging.
2. Exclusive input faults—those that require a transition on any input pin other than the one specified. An example of this fault is a break in wire segment p. To test for this fault, the output node must first be precharged with a stimulus from the set [(X, 0, X), (0, X, X)]. Then the stimulus (1, 1, 1) must be applied to discharge the output. If the gate to device 6 is floating at a value that does not permit 6 to turn on and stay on, the fault will be detected. (In other words, if this fault is dc-detectable at the time of the test, this test will detect it.)
3. Output faults—those that require a transition on the output pin. An example of this fault for a 0 to 1 transition is a break in wire segment v. The test for this fault is to apply a (1, 1, 1) stimulus to discharge the

Table 6 CMOS pin fault decision table.

CMOS pin fault	*Time 1*			*Time 2*		
	I	*¬I*	*T*	*I*	*¬I*	*T*
IsNC	CV/	/NC	CV/NC			
OsCV	NC/	NC/	NC/CV			
OsNC	CV/	CV/	CV/NC			
EtNC	X/X	CV/CV		NC/	NC/	NC/CV
ItCV	NC/NC	NC/NC		CV/	/NC	CV/NC
OtNC	CV/CV	CV/CV		NC/	NC/	NC/CV
OtCV	NC/NC	NC/NC		CV/	CV/	CV/NC

output and then apply a 0 to any input in an attempt to charge the output. An example of this fault for the 1 to 0 transition is device 4 open from source to drain. The test for this fault is to apply a 0 to any input to charge the output and then a (1, 1, 1) stimulus in an attempt to discharge the output.

These new definitions result in **Table 6,** the CMOS analog to the stuck-fault decision table (Table 3). The interpretation given to this table is the same as before. In addition, the X/X symbol means "assign any logic value" to both the GM and the FM. This is different from a blank, which means "do not assign anything." Consequently, the exclusive input transition fault requires two independent sets of decisions, the first to set the specific pin and the other to set all the other pins.

The ETG TSXXX function for the primitive blocks, TSAOI, implements this table. The code defining the objective in the BUGLIST data structure includes the CMOS faults. TSAOI assigns values for a given time image in the general way the stuck-fault TSAOI function does. The difference is that it does so for two time images on the input nets: two NETIMAGE elements are generated for these nets.

• *Transmitting responses to POs or SRLs*
The TRXXX functions are the same as for stuck faults. Note that TRXXX will be called for the second time image only.

• *Justifying stimuli back to PIs or SRLs*
For the standard primitives, the JUAOI function is the same as for the stuck-fault test generator. Of course, two images must be processed. However, the justify function for the SRLs, JUSRL, is fundamentally different from the stuck-fault version.

The basic justify philosophy implemented at an SRL is this: The first time image of a test is loaded into the appropriate SRL during the SRL load of the tester loop; the second time image is generated either by shifting the value in from the previous SRL in the scan string or clocking the value in through the data input.

To accomplish this, JUSRL must make a decision between the following two alternatives:

1. Generate an AB clock pair. Place the second time image value into the previous SRL in the first time image.
2. Generate a CB clock pair. Place the second time image value on the SRL's data input net in the first time image.

JUSRL must keep track of the decisions it makes.

• *Construction and validation of LSSD test patterns*
Tester loops generated by ETG for CMOS faults have a different format than the tester loops generated for stuck faults:

1. Load the SRLs.
2. Apply stimuli to the PIs.
3. Do one of the following:
   - Apply stimuli to the PIs.
   - Apply an AB clock pair.
   - Apply a CB clock pair.
4. Apply a CB clock pair.
5. Measure the PO responses.
6. Unload the SRLs.

The addition of the extra stimuli or clock pairs in the middle of the tester loop produces the sequences necessary to test the CMOS open faults.

CMOS fault simulators

The algorithms associated with the fault simulators used by ETG are not trivial. These simulators have two novel capabilities that are important to this discussion. Previous treatments of CMOS open failures [11] reflected that the nature of these defects implied a latch in the fault model that does not exist in the good machine model. The ETG simulators are capable of treating each block in the circuit as if such latches existed in the model.

The second capability involves a condition designated as a *B fault*. A B fault is a CMOS open fault in which the first time image setup is invalidated by a race condition that permits a momentary glitch to occur on a block pin other than the one under test [13]. In these simulation algorithms, each transition on a net goes through an unknown state ("*X*") prior to achieving the steady state value. This built-in pessimism effectively negates all effects of momentary glitches, thus suppressing implied tests on these B faults.

Experience with ETG

ETG is being used for test pattern generation by several IBM CMOS masterslice design systems. As part of the technology support, a library of fault models is provided, with an entry for each circuit, or "book," that can be placed on the masterslice. These models were developed using the following methodology:

1. The set of test objectives is determined for each circuit by determining the behavior of the circuit for each circuit fault. (The assumed failing modes are open metal or poly wire and open and shorted devices from source to drain.)
2. A logic circuit of primitive blocks is defined that the technologist believes will model both the circuit's GM and FM machine behavior.
3. CMOS pin faults are identified for each objective. That is, for each test objective, a CMOS pin fault is found in the logic model that generates the same tests as defined by the test objective. If such a fault cannot be found, three courses of action are possible:
   a. A new fault model can be defined that has an appropriate CMOS pin fault.
   b. The circuit can be redesigned so that that test objective is no longer required.
   c. The circuit can be identified as not fully testable and a quality detractor computed for each chip containing the circuit.

   In practice, all three courses of action have been taken.
4. ETG is run against the model to verify that the correct model has been defined and that the CMOS pin faults generate the predicted patterns.
5. Up to this point, the fault model contains both the pin faults that have been identified as necessary and all other possible pin faults on all the primitives in the model. If desired, the number of faults in the fault model can be reduced according to the following criteria:
   - Some CMOS pin faults on the primitives do not lead to legitimate test patterns at the inputs of the circuit. Assuming that these faults are not required to test the circuit, they can be eliminated from the model.
   - Other CMOS pin faults result in legitimate test patterns, but they are not required to test the fault modes that have been defined (for example, wire-to-wire shorts). If these fault modes are determined to be unimportant, the faults can be eliminated.

The design system procedure for test generation starts with a data set defining the design in terms of a design language. The basic operators in this language are the books of the design system. This data set is processed in order to obtain a preliminary logic structure on which LSSD design rules checks are run, the stability state defined, and the shift register tests generated. In the next step, a more refined logic structure is generated, the LMOD, along with the BUGLIST. Finally, a test generation system is called that, in turn, uses ETG to generate the test patterns and one of ETG's supporting simulators to determine the test coverage.

Statistics on a few designs processed by ETG are shown in **Table 7.** The *Blocks* column shows the number of blocks in the design, *SRLs* the number of SRLs. *SRL %* is the approximate percent of the system's blocks devoted to SRLs; *Come froms* is the number of block inputs in the system—this number gives an idea of the design's complexity; *Stuck faults* is the number of stuck faults; *CMOS faults* is the number of CMOS open pin faults. (From this point on, *CMOS faults* will always mean CMOS open pin faults, unless otherwise indicated.)

The S1 design in this table is a simple but important circuit. It is the circuit used by Reddy et al. [12] as an example of a circuit with a B fault.

S2 is another special design. It has a four-bit counter that has proven particularly resistant to CMOS testing.

The other designs are chips that have been designed for product applications. Note that S4 is mostly latches.

Note also that the total number of faults using a CMOS fault model is about double the number of faults in a stuck-fault model.

Table 8 is a comparison of the fault coverage provided by ETG and a popular IBM stuck-fault test generator. The columns in the table are the following:

Stuck fault	Data from the stuck-fault generator with no shift register test fault simulation before the stuck-fault generator was run.
Stuck fault + SR	Data from the stuck-fault generator with the shift register test fault simulated before the stuck-fault generator was run.
ETG	ETG data without shift register test fault simulation.
ETG + SR	ETG data with shift register test fault simulation.
SF	Percent test coverage on the stuck faults.
CM	Percent test coverage on the CMOS faults.
T	Percent test coverage for all faults.

Table 7 Statistics of some CMOS designs.

Chip	*Blocks*	*SRLs*	*SRL %*	*Come froms*	*Stuck faults*	*CMOS faults*
S1	15	0	0	20	45	50
S2	186	4	60	324	381	430
S3	1179	24	57	1976	2183	2278
S4	4837	144	83	9095	7866	8230

Table 8 Percent fault coverage for some CMOS designs.

Part	*Stuck fault*			*Stuck fault + SR*			*ETG*			*ETG + SR*		
	SF	*CM*	*T*	*SF*	*CM*	*T*	*SF*	*CM*	*T*	*SF*	*CM*	*T*
S1	100	0	47	—	—	—	100	98	99	—	—	—
S2	93	30	61	93	41	68	96	82	89	96	82	89
S3	88	31	44	89	53	72	94	76	85	95	78	86
S4	95	36	65	95	55	75	97	70	83	99	73	86

On the basis of the data in Table 8, as well as other data, some conclusions can be drawn about the efficacy of stuck-fault LSSD ADTPGs and ETG:

1. ETG and the fault simulator do work correctly in not indicating a successful test for B faults.
2. CMOS fault coverage by the stuck-fault LSSD ADTPG without the shift register test is uniformly low. CMOS fault coverage of less than 40 percent should be expected. The reason for this is obvious when the tester loops are examined. There is a low probability that the correct sequence of clocks will be generated to provide the test sequences necessary to catch the CMOS faults.
3. The shift register test can provide a significant increase in CMOS fault coverage over that provided by an LSSD ADTPG alone. The greater the proportion of SRLs to combinational logic in the design, the better the test coverage. This is because the shifting operation produces useful sequences for the SRLs and logic around the SRLs.
4. ETG provides a significant increase in the CMOS fault coverage over the stuck-fault ADTPGs, even with a shift register test. However, the CMOS fault coverage, which is in the 75-percent range, is lower than the stuck-fault coverage for the design. Studies are underway to determine how the coverage might be improved.

Several performance criteria were listed in the Introduction section for evaluating ADTPGs. As demonstrated above, if the measure used for the "percentage of fault machines detected" criterion is the percent fault coverage, ETG has significantly better performance than the stuck-fault ADTPG.

Data on the other criteria, namely 1) time required to generate test patterns, 2) CPU resource requirements, 3) test time, and 4) shipped product quality level are not included in this paper either because development of some aspects of ETG is still underway or there are insufficient data.

Nevertheless, some general qualitative comments about the expected performance of ETG, compared to a stuck-fault ADTPG, are possible:

- ETG's "shipped product quality level" should be better. Of course, the actual improvement depends, in part, on the maturity of the technology: Older technologies should see less improvement than younger technologies simply because the yield on older technologies is higher.
- ETG should require more elapsed time for test generation and more computer resources. Based on a doubling of the number of faults and a doubling of the number of time images that need to be processed, a crude estimate of a factor of four increase in these criteria might be expected.
- The time required to apply ETG-generated patterns at the tester is expected to increase. The magnitude of this increase depends, to a significant degree, on the number of patterns, the design's architecture, and the tester. The actual number of tester stimuli should be expected to be at least approximately two times that required for a stuck-fault test. This lower bound is achieved if a minimum-size set of patterns to test the block input stuck faults can be modified by separating succeeding patterns with preconditioning patterns [13]. For example, the patterns that test a three-input AND block are (011, 101, 110). The minimum-size CMOS fault test pattern set is (111, 011, 111, 101, 111, 110).

Summary

This paper introduces the topic of automatic test pattern generators by describing in abstract terms the systems that they process. The three major types of automatic test pattern generators are defined. ETG is an ADTPG. Its high-level structure is presented. The stuck-fault concept is reviewed and then the operation of ETG stuck faults is discussed, with emphasis placed on the Test, Justify, and Transmit functions.

Building on this base, new CMOS pin faults are defined and ETG operation with respect to these faults is described, again with emphasis on the Test, Justify, and Transmit functions. The fundamental capability of ETG is to generate the sequences of length 2 necessary to detect the CMOS technology circuit faults that can be detected by a static test. It does this by using extra scan clocks or system clocks.

Some data are presented comparing the performance of ETG and a stuck-fault ADTPG. It is concluded that the LSSD stuck-fault algorithms alone are not effective against CMOS faults, that shift register tests can help, and that ETG does significantly increase the CMOS fault coverage.

Acknowledgments

P. Shearon was the project leader for ETG development. K. McCauley and E. Ciechoski in Endicott and D. Leet in Burlington helped with the coding. R. France developed the CMOS pin fault concept, with help from D. Leet. W. Maloney, in Endicott, was the project leader for the CMOS fault simulator development. He and R. France developed the basic algorithms. K. Melocco, in Endicott, helped with the coding. D. Leet, L. Thorsen, E. Bouldin, and P. Moritz, all in Burlington, developed the technology fault models and helped test the system.

References

1. T. W. Williams and K. P. Parker, "Design for Testability—A Survey," *IEEE Trans. Computers* **C-31,** 2–15 (1982).
2. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," *Proceedings of the 14th Design Automation Conference*, New Orleans, 1977, pp. 462–468.
3. T. Guzowski, "Automatic Test Generation for VLSI," *Proceedings of the European Conference on Electronic Design Automation*, 1981, pp. 227–230.
4. M. Murakami, N. Shiraki, and K. Hirakawa, "Logic Verification and Test Generation for LSI Circuits," *Proceedings of the 1980 IEEE Test Conference*, pp. 467–472.
5. C. Bellon, Ch. Robach, and G. Saucier, "An Intelligent Assistant for Test Program Generation: The 'Supercat' System," *Proceedings of the IEEE International Conference on Computer-Aided Design*, 1983, pp. 32–33.
6. M. A. Breuer et al., "State-of-the-Art Assessment of Testing and Testability of Custom LSI/VLSI Circuits, Vol IV: Test Generation," *Technical Report SD-TR-83-20*, U.S. Air Force Space Division, Los Angeles, 1982.
7. J. Grason and A. W. Nagle, "Digital Test Generation and Design for Testability," *J. Digital Syst.* **5,** 319–359 (1981).
8. P. Goel and B. C. Rosales, "PODEM-X: An Automatic Test Generation System for VLSI Logic Structures," *Proceedings of the 18th Design Automation Conference*, 1981, pp. 260–268.
9. R. D. Davies, "The Case for CMOS," *IEEE Spectrum* **20,** 26–32 (1983).
10. R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *Bell Syst. Tech. J.* **57,** 1449–1474 (1978).
11. R. L. Wadsack, "Fault Coverage in Digital Integrated Circuits," *Bell Syst. Tech. J.* **57,** 1475–1488 (1978).
12. S. M. Reddy, M. K. Reddy, and J. G. Kuhl, "On Testable Design for CMOS Logic Circuits," *International Test Conference Proceedings*, 1983, pp. 435–445.
13. Y. M. Elziq, "Automatic Test Generation for Stuck-Open Faults in CMOS VLSI," *Proceedings of the 18th Design Automation Conference*, 1981, pp. 347–354.

Received December 1, 1983; revised March 13, 1984

R. France *IBM General Technology Division, P.O. Box 6, Endicott, New York 13760.* Mr. France is an advisory engineer working in test generation development in the Engineering Design System (EDS), where he is responsible for developing the models used in test generation. He received the B.E. in electrical engineering from Youngstown State University, Ohio, in 1967 and the M.S in electrical engineering from Syracuse University, New York, in 1972. Mr. France joined IBM in Endicott in 1967.

D. Leet *IBM General Technology Division, Burlington facility, P.O. Box A, Essex Junction, Vermont 05452.* Dr. Leet is a staff engineer in the Test Generation Department and is responsible for developing test generation methodologies and tools for Burlington laboratory logic products. He received the B.S. in electrical engineering in 1966, the M.S. in biophysics in 1968, and the Ph.D. in systems science in 1971, all from Michigan State University, East Lansing. In 1971 he joined the faculty of the University of Evansville, Indiana, where he taught computing science. From 1974 to 1978, he was on the staff of the University of Dayton Research Institute, Ohio, where he directed projects in speech recognition, biomechanics, and laser system data acquisition and control. Dr. Leet joined IBM in Burlington in 1978.

P. Shearon *IBM General Technology Division, P.O. Box 6, Endicott, New York 13760.* Mr. Shearon is a staff engineer in the Logic Test Generation Development Department of test design automation and is responsible for developing test generation tools for use in the Engineering Design System. He received a B.S. in mechanical engineering from the University of Toledo, Ohio, in 1964 and an M.S. in computer systems from the State University of New York in 1976. Mr. Shearon joined IBM in Endicott in 1964.