E Computer Society
0662 Los Vaqueros Circle
O Box 3014
os Alamitos, CA 90720-1264
hone: (714) 821-8380
ax: (714) 821-4010
.e.english@compmail.com

IEEE
Design&Test of Computers

Spring 1994

Volume 11 Number 1

FEATURES

ARTICLES

MULTICHIP MODULES, PART 2

ALSO IN THIS ISSUE

DEPARTMENTS

Design Validation:

Comparing Theoretical and Empirical Results of Design Error Modeling

SUNGHO KANG
Motorola Inc.
STEPHEN A. SZYGENDA
University of Texas at Austin

To use simulation for design verification, designers need a confidence measure for a given set of simulation patterns, specifically for cases in which only a subset of the possible patterns is used. The authors derive a measure of design verification coverage based on the number of design errors detected in a theoretical analysis of a circuit. To verify the theoretical analysis, they simulate errors and compare the results.

SIMULATION IS A common method for verifying the design of digital systems. But as systems increase in size and complexity, simulating them becomes very costly. So system designers generally use only a subset of all the possible simulation patterns for verification. Because a measure of completeness is not available except for exhaustive simulation, the question arises as to how much of the design has been verified. To determine the extent of design verification, designers need a measure of simulation coverage for a given set of simulation patterns.

Presently, simulation coverage (SCP) is often represented as the number of simulated patterns divided by the number of possible patterns times 100%. However, this conventional measure does not accurately answer the design verification question because it assumes that the probability of each design error and pattern is uniformly distributed.

To provide more realistic coverage, we introduce a simulation coverage metric[1] based on the number of modeled errors. Since this metric does not assume a uniformly distributed pattern, it provides more realistic results than those presently available. Such a metric can have a profound impact on the entire design validation process.

Theoretical analysis

We derive a theoretical simulation coverage by computing the number of design errors in a circuit on the basis of certain assumptions. Let n be the number of primary inputs, u the number of primary outputs in the circuit, and s the number of state variables. Then, $F = 2^{2^{n+s}}$ functions are possible, if we consider only 0 or 1 as inputs. Let $S = 2^{n+s}$ be the number of possible simulation patterns that exercise all primary input value combinations and all state variable value combinations. Let the possible functions be $f_1, f_2, \ldots, f_F$, and let $f_j(i)$ denote the value of the function f_j when simulation pattern i is applied.

We define a *design error* as a design change that causes a different output value from the desired value for some input pattern. A design change that is functionally equivalent to the output of the desired function is not an error.

The definition can be represented formally. Consider a design change α. Assume that under this change, function f_j changes to f_j^{α}. We formulate it as

0740-7475/94/$04.00 © 1994 IEEE

$f_j(i) \oplus f_j^{\alpha}(i) = 1$. Generally, design change α becomes a design error when

$$\sum_{i=1}^{S}\left[\sum_{j=1}^{F} f_j(i) \oplus f_j^{\alpha}(i)\right] = 1$$

This equation implies that if any output value for any input vector differs from the desired value, we call the change a design error. Then, we can represent the simulation coverage as the number of detected errors divided by the number of errors times 100.

Under this definition of a design error, the number of errors in two circuits can be the same, although the circuit structures are different, as shown in Figure 1. This implies that for the same function, the number of design errors is the same. Therefore, the number of design errors is functionally, not structurally, dependent.

If a circuit has more than one output, we assume that the outputs function independently of one another. Thus, the total number of design errors in a circuit is the sum of the design errors that we can detect at each output.

Combinational circuits. Let E_l^u be the set of errors detected at output u by pattern l, and let $\| E_l^u \|$ be the magnitude of E_l^u. Since there are no state variables, $F = 2^{2^n}$ and $S = 2^n$. We represent the number of errors detected by pattern i as

$$\sum_{k=1}^{F} f_k(i) \oplus f_m(i)$$

where the desired output is f_m. Also, the number of errors detected by patterns i and j are

$$\sum_{k=1}^{F}\left[\left(f_k(i) \oplus f_m(i)\right)\left(f_k(j) \oplus f_m(j)\right)\right]$$

The set of detected errors in a circuit is the union of the sets of detected errors for each output:

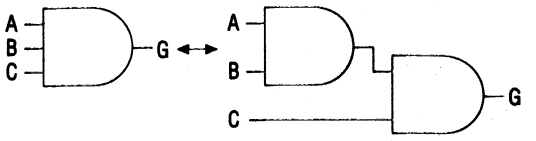


***Figure 1.** Circuit structure independence.*

$$E = \cup_{i=1}^{u} E^i = \cup_{i=1}^{u}\left[\cup_{j=1}^{l} E_j^i\right]$$

Let q_i be the number of primary inputs connected to the primary output i, where $1 \le q_i \le n$. Then, there are $2^{2^{q_i}} - 1$ errors for i. Therefore, the total number of errors in the circuit is

$$\sum_{i=1}^{u}\left(2^{2^{q_i}} - 1\right)$$

Thus, to detect all detectable errors at output i requires 2^{q_i} of $S = 2^n$ patterns. Without loss of generality, let the inputs related to output i be $j_1, j_2, \ldots, j_{q_i}$. For example, let the first pattern be $j_1 = 1$, $j_2 = 1, \ldots, j_{q_i} = 1, \ldots, j_n = 1$; let the other pattern be $j_1 = 1, j_2 = 1, \ldots, j_{q_i} = 1, \ldots, j_n = 0$. Let both patterns produce the same output at i. Furthermore, assume that when the second pattern is applied, no new design errors other than those detected by the first pattern are detected at output i. Therefore, the second pattern is redundant for testing at output i.

We define the number of distinguished patterns at output i as $d_i(l)$, which is the number of nonredundant patterns at output i, among the applied l patterns. Therefore, in the example, $d_i(l)$ equals 1 for the first pattern, and $d_i(l)$ equals 1 for the second pattern, since the second pattern is redundant to output i. The boundary for d_i becomes $0 \le d_i(l) \le 2^{q_i}$.

To easily compute the number of detected errors with l patterns, we introduce a W function. Let $W(x,y)$ be the number of errors detected by y patterns among x patterns. Then, $W(x,y) = 2^{x-1} + 2^{x-2} + \ldots + 2^{x-y}$. Here, $W(x,0) = 0$; $W(x,1) = 2^{x-1}$; $W(x,x) = 2^{x-1}$; $W(x,y) = W(x,y-1) + 2^{x-y}$.

Since the number of errors detected at the primary output i, using l patterns, is $W(2^{q_i}, d_i(l))$, the total number of errors detected by l patterns is

$$\|E\| = \sum_{i=1}^{u} W\left(2^{q_i}, d_i(l)\right)$$

and the simulation coverage is

$$\sum_{i=1}^{u} \frac{W\left(2^{q_i}, d_i(l)\right)}{\left(2^{2^{q_i}} - 1\right)} \times 100\ (\%)$$

Sequential circuits. Asynchronous sequential circuits can be handled similarly to combinational circuits. However, synchronous sequential circuits are more complex because they involve state variables. For this case, let q_i be the number of primary inputs and state variables connected to an output i. To compute $\| E \|$, consider an infeasible state to be a logic value that a state variable cannot assume. Let r_i be the number of infeasible state values for state variables connected to output i. The existence of r_i implies that the number of errors in the circuit decreases because some functions will not be observable at the output.

Therefore, the number of errors detected at the primary output i, using l patterns, is $W(2^{q_i - r_i}, d_i(l))$, where $0 \le d_i(l) \le 2^{q_i - r_i}$. The total number of errors detected by l patterns is

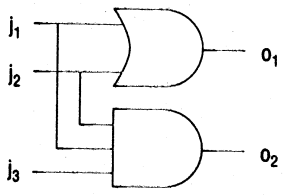


Figure 2. Example circuit.

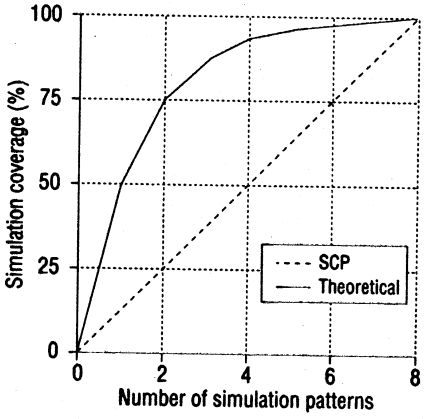


Figure 3. Theoretical simulation coverage and SCP for circuit in Figure 2.

$$\|E\| = \sum_{i=1}^{u} W\left(2^{q_i - r_i}, d_i(l)\right)$$

The simulation coverage is

$$\sum_{i=1}^{u} \frac{W\left(2^{q_i - r_i}, d_i(l)\right)}{\left(2^{2q_i - r_i} - 1\right)} \times 100\ (\%) \qquad (1)$$

This is a general simulation coverage equation for any circuit.

Conclusions of the analysis. Obviously, achieving 100% simulation coverage requires all possible patterns in simple output cases, if we ignore possible redundancy. However, theoretical analysis has shown that the number of patterns is not proportional to the simulation coverage. For example, consider a circuit with three inputs and two outputs, as shown in Figure 2. For convenience, let o_1 and o_2 be the two outputs and j_1, j_2, and j_3 be the three inputs. If o_1 is related to the two inputs j_1 and j_2, and o_2 is related to all three inputs, q_1 equals 3, and q_2 equals 2. The total number of errors in this circuit is

$$\sum_{i=1}^{u} \left(2^{2^{q_i}} - 1\right) = \left(2^4 - 1\right) + \left(2^8 - 1\right) = 270$$

As the number of applied patterns increases, consider the design error coverage. Table 1 shows the number of detected errors. Figure 3 shows the theoretical simulation coverage and SCP prediction.

We can conclude that the actual coverage, represented by simulation, is higher than we thought. Our theoretical analysis indicates that a metric based on the number of design errors provides more accurate coverage than SCP and that the conventional assumption of uniform distribution of patterns is inaccurate. Therefore, the choice of simulation patterns is important. An ideal way to achieve efficient design verification is to choose a proper set of simulation patterns by using the previous analysis. However, deriving $\|E\|$ is almost impossible for large circuits, even though we performed the analysis without considering timing. If we considered timing, the analysis would be more complex.

Table 1. The number of detected errors for circuit in Figure 2.

Pattern	Detected errors at output 1	Detected errors at output 2	Total errors
000	8	128	136
001	8	192	200
010	12	224	236
011	12	240	252
100	14	248	262
101	14	252	266
110	15	254	269
111	15	255	270

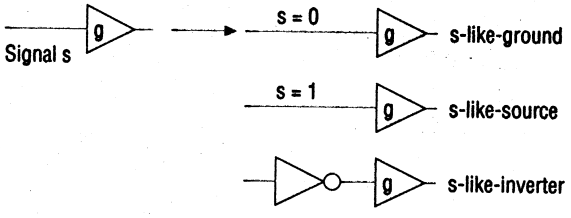


Figure 4. Signal errors.

New design validation approach

Because considering all possible design errors for large circuits is almost impossible, we must find a method of modeling only the design errors most likely to occur. The approach we propose uses a new simulation coverage metric based on design error modeling with reasonable assumptions:

$$\text{SCM} = \frac{\text{number of detected errors}}{\text{number of modeled errors}} \times 100\ (\%) \qquad (2)$$

Design error models. By modeling design errors, we can derive a mea-

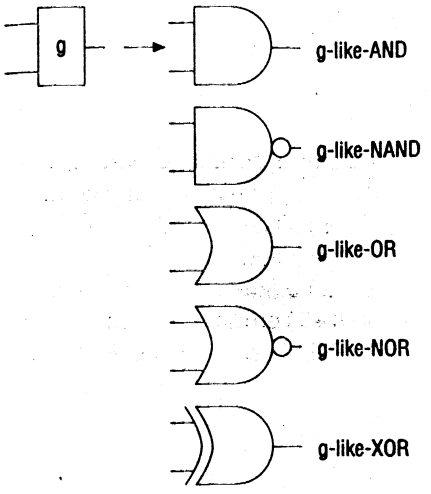


Figure 5. Gate errors.

surement of design validation confidence. In our research, we considered only gate-level designs, but this approach can be extended to higher level functional elements. Since the modeling of all possible errors is too complex and the number of errors is too large, we consider only a subset of all possible errors. We use several design error models: *signal, gate, local,* and *include.*

The first model is an error related to a signal. Let *s* be the signal. In an *s*-like-source error, signal *s* acts like a source. In an *s*-like-ground error, signal *s* acts like a ground. An *s*-like-inverter error acts like the inverted signal of *s*.

A *gate error* is related to a gate function. Let *g* be a gate in the circuit. In a *g*-like-AND error, *g* works like an AND gate. In a *g*-like-OR error, *g* works like an OR gate. In a *g*-like-NAND error, *g* works like a NAND gate. In a *g*-like-NOR error, *g* works like a NOR gate. In a *g*-like-XOR error, *g* works like an XOR gate. Examples of signal and gate error models are shown in Figures 4 and 5.

A *local error* has two error sites represented by only one gate error and one signal error, where the signal is connected to the gate. Figure 6 shows an example of a local error.

Next we consider design errors involving a change in the number of inputs to a gate. First, consider errors that increase the number of inputs. Figure 7 shows an example circuit. Initially, gate B has two inputs. Subsequently, because of a design error, gate B has three inputs. This error cannot be modeled as a signal or gate error. To represent this error, we introduce a new error model, the *include error.* Let *s* be a signal and *g* be a gate. A *g*-include-*s* error occurs when gate *g* includes the signal *s* as an input, though *s* is not intended to be an input of *g*.

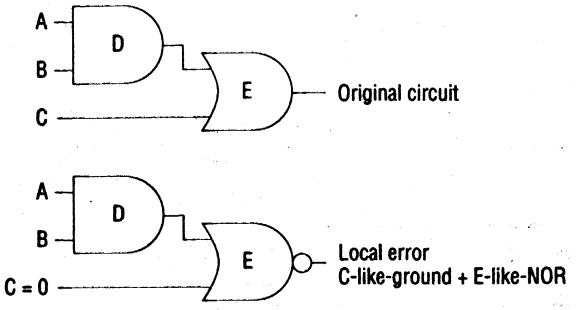


Figure 6. Example of a local error.

To model errors that decrease the number of inputs, we use signal errors, as shown in Figure 8, next page. For AND or NAND gates, we model a change in the number of inputs with *s*-like-source. For OR or NOR gates, a change in the number of inputs can be modeled with *s*-like-ground. For XOR gates, we model the error with *s*-like-ground.

These error models represent a subset of all possible design errors. Individual users may wish to exclude some errors or add others that are appropriate to their design environments. Users can easily model specific errors and add them to the error set according to their experience and knowledge of the design.

Assumptions. Our objective is to model design errors that are the most likely to occur. Therefore, we limit the number of design errors to be modeled by introducing the following reasonable assumptions.

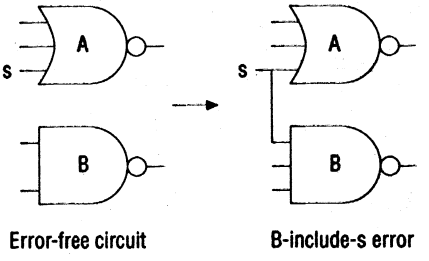


Figure 7. Example of an include error.

Single-error assumption. We assume a single error is a design error that can be modeled by a signal error or a gate error. In the circuits to be considered, we use single-error models.

This assumption is analogous to the classical single-fault assumption extensively used for fault analysis.[2] (A single-fault model covers most multiple faults in a simulation environment, and this is also likely for design errors.) The single-error assumption provides a reasonable approximation, since most design errors are related to misuse of basic primitives. For example, consider the desired function $f = AB + C$ shown in Figure 9. We can model a design error that changes the function to $f' = A + B + C$ as *D*-like-OR.

If we use the single-error assumption, we need consider only a small portion of all errors. We can model a subset of

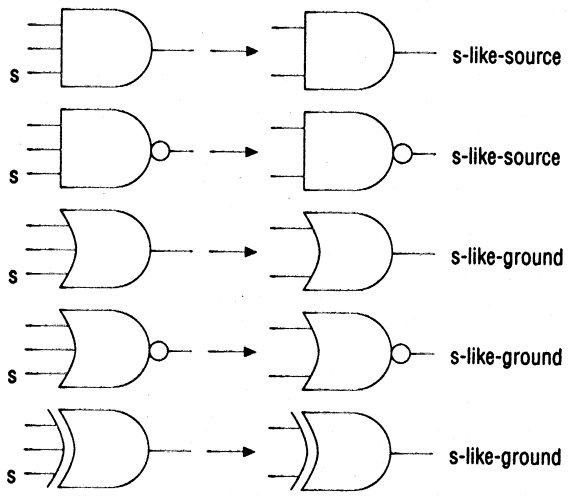


Figure 8. *Errors of input number decrease.*

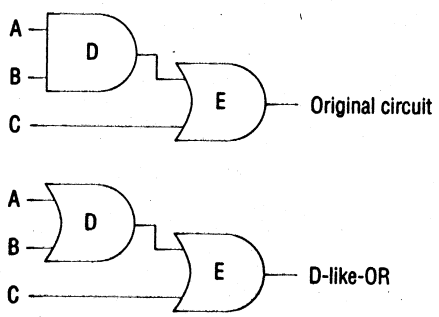


Figure 9. *Example of single-error assumption.*

errors using the single-error assumption, but some errors cannot be modeled as signal and gate errors. Therefore, the obvious problem with the single-error assumption is that it does not model some errors of concern. We can model these errors using a multiple-error concept, by making the following assumption.

Local-error assumption. We consider only design errors that can be modeled by local-error models. This assumption is reasonable since many design errors occur in a neighborhood. For example, consider the desired function $f = AB + C$, shown earlier in Figure 6. We can model a design error that changes the function to $f' = \overline{ABC}$ as *D*-like-NAND + *C*-like-inverter.

There are still many errors of interest not modeled under the preceding assumptions. To consider these, we introduce the following assumption.

Include-error assumption. We consider design errors that can be modeled as include errors.

Based on the three assumptions, we can generate an equation describing the total number of design errors. Before doing so, we must define equivalent errors. In a circuit whose behavior is represented by a function f, let's define the set of simulation patterns that detects an error α as $S_\alpha = f \oplus f_\alpha$. We define the set of simulation patterns that detects an error β as $S_\beta = f \oplus f_\beta$. We define the set of simulation patterns that distinguishes α and β as $f_\alpha \oplus f_\beta$. If f_α equals f_β, no simulation patterns distinguish α and β. Such errors are said to be equivalent and can be represented by one error. *EQ* represents the total number of equivalent errors in a circuit.

Consider a circuit consisting of k gates with n primary inputs. The number of possible errors is the function of the number of primary inputs (2^{2^n}). For a gate with p fan-ins, we have at most $4 + 3 \times (p + 1)$ errors. The number of errors modeled using the previous three assumptions is $4k + 15(P + k) + k(P + k) - EQ$, where

$$P = \sum_{i=1}^{k} p_i$$

In the equation, the first term represents the number of gate errors, the second term represents the sum of the number of signal errors and the number of local errors, and the third term represents the include errors.

Error simulation. Error simulation is the process of simulating a circuit to analyze its operation under various design error conditions. Error simulation enables us to evaluate the error detection effectiveness of simulation patterns by measuring SCM (Equation 2).

Figure 10 shows the error simulation algorithm. The inputs to the simulation are a circuit description and a set of simulation patterns. The simulator translates the circuit description into internal tables and generates the error list according to a circuit topology. The error list includes the position and type of each error. All errors are assigned a level and stored in the list according to level. Multiple errors, which have more than one error site, are stored according to the lowest level error site. The simulator considers error collapsing[3] to minimize the size of the error list. Error collapsing is the process of representing equivalent errors as a single error for simulation.

After creation of the error list, the circuit is simulated without any design er-

rors. To do this, the simulator selects 32 patterns and simulates them in parallel, storing the results of the evaluation for comparison. Then, the simulator selects an error from the error list and executes an error model replacement—a procedure to change the structure of a circuit according to an error. This procedure takes place after selection of a target error.

When the simulator encounters an error site specified in the error list, it evaluates the error model and checks for propagation of the error effect. If the value of the evaluation is the same as the value of error-free simulation, the error effect has not propagated, indicating that this error cannot be detected. Therefore, simulation stops, another error is selected, and simulation restarts. However, if the target error is a multiple error, the simulation continues until the last error site is considered. If the error effect propagates to any of the primary outputs, the error is detected and removed from the error list. The number of detected errors is then recorded. The procedure continues until all errors are detected or all the given patterns are simulated. Finally, the simulator provides the SCM.

To handle any type of design errors, we have implemented parallel and concurrent design error simulators.[4]

```
create an error list
for all patterns
        choose 32 patterns that are not simulated
        simulate a circuit without any error
        store the values
        for all errors that are not detected
                select an error which has lowest level
                execute an error model replacement
                simulate a circuit
                        if the value is the same as the value in the
                                previous error-free simulation
                                after highest error site evaluation
                        stop simulation for this error
                detect an error
                if this error is detected
                        delete the error from the error list
                        count the number of detected errors
        end
end
```

***Figure 10.** Error simulation algorithm.*

***Table 2.** Gate error excitation table for an OR.*

Gate error	Excitation	Value
G-like-AND	At least one 0 and 1 of fan-ins	E
G-like-NAND	All 0's of fan-ins	$\bar{E}$
	All 1's of fan-ins	E
G-like-NOR	All 0's of fan-ins	$\bar{E}$
	At least one 1 of fan-ins	E
G-like-XOR	All 1's and the number of 1's is even	E

Automatic error pattern generation. In error simulation, the selection of efficient simulation patterns is important, since each pattern can detect a different number of design errors. Also, to detect a specific design error, the simulator requires a specific simulation pattern. To accomplish this, we have developed an automatic error pattern generation (AEPG) program, which is analogous to automatic test pattern generation.[5,6] The main difficulty of AEPG is that there are many possible ways to excite the error effect. In our AEPG program, we use the following extra logic values to represent the error excitation effect: The value E represents 1 in an error-free circuit and 0 in an erroneous circuit. The value $\bar{E}$ represents 0 in an error-free circuit and 1 in an erroneous circuit.

First, consider signal error excitation. To excite a signal-like-ground error, the signal must be set to 1. For a signal-like-source error, the signal must be set to 0. However, to excite a signal-like-inverter error, the signal can be set to 0 or 1.

For each primitive, we use gate and local error excitation tables to excite the signal. Let G be the name of a gate, Z the output signal of the gate, and A the input signal of the gate. Tables 2 and 3 show the gate error excitation table and the local error excitation table for an OR gate. For local errors, both error sites must be excited at the same time.

Consider the excitation of an include error. It is more difficult to excite than the other errors because it may generate a reconvergent fan-out in the circuit. If the include error does not generate a new reconvergent fan-out, excitation of the include error is similar to local-error excitation. However, if the include error does generate a new reconvergent fan-out, error excitation is more complex.

Table 3. Local error excitation table for an OR.

Local error	Excitation	Value
G-like-NOR + A-like-ground	All 0's of fan-ins	$\bar{E}$
	At least one 1 of other fan-ins	E
G-like-NOR + A-like-inverter	At least one 1 of other fan-ins	E
G-like-XOR + Z-like-inverter	All 0's	$\bar{E}$
	The number of 1's is odd	E

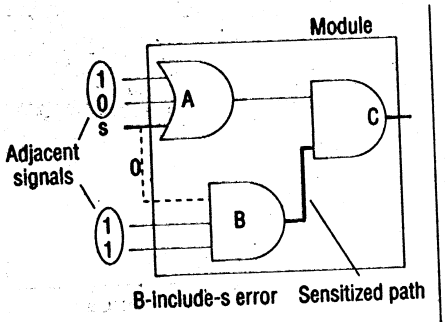


Figure 11. Include-error excitation example.

To excite an include error that creates a new reconvergent fan-out, we introduce a module and adjacent signals. G is the gate and S is the signal; the error is G-include-S. We define the *module* as a set of gates that includes gates along the path between a new reconvergent fan-out and a point of reconvergence due to an error. The *adjacent signals* are fan-ins of the gates in the module, except signal S.

To excite an include error, the path from the gate G to the point of reconvergence must be sensitized. For this to occur, all adjacent signals must be set to nondominant values (1 for AND and NAND and 0 for NOR and OR); the new input S of gate G must be set to a dominant value (0 for AND and NAND and 1 for NOR and OR). An example appears in Figure 11.

Figure 12 shows the AEPG algorithm. In AEPG, an error can be set to E or $\bar{E}$ at an error site. Therefore, after considering excitation for setting an error to E, if an error simulation pattern is not generated, the pattern generator executes new excitation by setting an error to $\bar{E}$. If the pattern generator observes an error effect at the primary outputs, it regards the error as detected and generates a new pattern. Using the new pattern, the error simulator executes simulation to detect other design errors, since an error pattern usually detects more than one design error.

Results

Using Equation 1, we could compute a theoretical simulation coverage for small circuits. However, as we have said, for large circuits, this computation is too large. The number of design errors is $O(2^{2^{n+s}})$, where n is the number of primary inputs and s is the number of state variables. Also, this theoretical analysis can be applied only in a zero delay environment. Thus, for a more practical measure, we implemented error simulation and AEPG, with the following results.

Error simulation. The error simulator handles both combinational and sequential circuits because a design error can change a combinational circuit into a sequential circuit, and vice versa. Table 4 shows error simulation results we obtained using a Sun Sparc 10 with 32 Mbytes of main memory and a swap space of 172 Mbytes. In this simulation, we used 1,000 random patterns for the benchmark circuits.[7,8] The set of modeled errors included single, local, and include errors related to primary inputs.

For combinational circuits, the SCMs were high, which means that many design errors were detected using these patterns. However, for sequential circuits, the coverage was quite low because random patterns were used.

The significance of our results is that the design error simulator not only tells the user that the coverage is low, it also indicates which parts of the circuits were not covered. This is the first error simulation system that provides this level of information. Furthermore, the error simulation results are very close to those predicted by theoretical analysis. The error simulation results verified the accuracy of the SCM, demonstrating that error simulation can provide a measure of simulation coverage for large circuits in a timing environment.

Design errors and faults. Our results do not imply that classical fault simulation can be generally used for any reasonable measure of design validation. However, note that fault models can be mapped into a subset of design error models because all fault models have corollaries in design error models. Thus, one can use our new design validation metric as a fault analysis metric, using a measure of the fault model subset. Consequently, fault assessment may become a by-product of design validation, with all the inherent efficiencies that could result from this approach.

Automatic error pattern generation. Table 5 shows the results of AEPG. The abort errors are errors that are not

```
while a certain coverage is derived do
    generate 32 random patterns
    execute error simulation using these patterns
    store the pattern which detects at least one error
end
for all undetected errors in the list
    if the error is an include error
        select a module and adjacent signals
    for all excitations of the error
        excite an error according to an error type
        propagate an error effect to primary outputs
        if an error pattern is generated
            mark the error detected
            store the generated patterns
            break
    if an error pattern is not generated
        delete this error from the list
    if the number of generated patterns is 32
        execute error simulation
end
```

Figure 12. AEPG algorithm.

Table 4. Error simulation results.

Circuit	Modeled errors	SCM (%)	SCM time (s)
c432	6,689	98.80	1.69
c499	9,172	99.61	3.01
c880	21,868	99.64	6.95
c1355	26,020	98.24	24.24
c1908	22,887	95.71	30.39
c2670	147,824	82.16	82.51
c3540	65,778	80.26	64.15
c5315	274,502	94.74	107.78
c6288	107,376	98.46	461.83
c7552	448,099	94.72	206.30
s641	5,483	38.51	3.68
s953	9,846	24.19	9.82
s1423	15,029	10.85	36.13
s5378	56,998	19.70	123.65
s9234	70,895	1.23	609.62

detected because the number of backtracks is limited. If we allowed unlimited backtracks, we could achieve a higher SCM and detect all abort errors. Additionally, the generation time would increase significantly.

Table 5. Error pattern generation results.

Circuit	Modeled errors	Abort errors	SCM (%)	Generation time (s)
c432	6,689	77	98.85	7.81
c499	9,172	38	99.56	9.38
c880	21,868	77	99.64	30.96
c1355	26,020	449	98.27	209.22
c1908	22,887	444	98.06	163.20
c2670	147,824	1,961	98.67	817.88
c3540	65,778	1,908	97.09	715.37
c5315	274,502	1,697	99.38	2,584.11
c6288	107,376	18	99.98	1,668.98
c7552	448,099	2,775	99.38	7,436.25

OUR RESEARCH LED TO A TOOL that can be used to measure the degree of design validation that has been achieved, can specify which areas of design have or have not been tested, and can generate tests for selected errors. Future work will involve the analysis of specific models of design errors, development of practical methodologies for the use of these models, and optimization of the simulation and test generation algorithms.

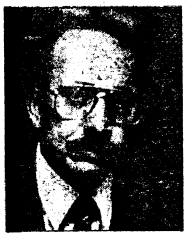
References

1. S. Kang and S. Szygenda, "New Design Error Modeling and Design Validation Metrics," *Proc. Third European Design Automation Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1992, pp. 471-477.
2. E. Thompson and S. Szygenda, "Digital Logic Simulation in a Time-Based, Table-Driven Environment: Part 2, Parallel Fault Simulation," *Computer*, Vol. 8, No. 3, Mar. 1975, pp. 38-49.
3. S. Kang and S. Szygenda, "Modeling and Simulation of Design Errors," *Proc. Int'l Conf. Computer Design*, IEEE CS Press, 1992, pp. 443-446.
4. S. Kang and S. Szygenda, "Comparative Analysis of Error Simulation Algorithms for Design Validation," *Proc. Modeling and Simulation Conf.*, Technical Com-

munication Service, Pittsburgh, 1992, pp. 2553-2564.
5. P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Computers*, Vol. 30, No. 3, Mar. 1981, pp. 215-222.
6. M. Schulz, A. Trischler, and T. Sarfert, "Socrates: A Highly Efficient Automatic Test Pattern Generation System," *Proc. Int'l Test Conf.*, IEEE CS Press, 1987, pp. 1016-1026.
7. F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *Proc. Int'l Symp. Circuits and Systems*, IEEE CS Press, 1985, pp. 695-698.
8. F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. Int'l Symp. Circuits and Systems*, IEEE CS Press, 1989, pp. 1929-1934.

Sungho Kang is a senior staff engineer at Motorola's Semiconductor Systems Design Technology Laboratory. Previously, he was a research scientist at Schlumberger Laboratory for Computer Science and a postdoctoral fellow at the University of Texas at Austin. His research interests include design verification, fault simulation, testing, and design for testability. Kang received the BS degree from Seoul National University, Korea, and the MS and PhD degrees in electrical and computer engineering from the University of Texas at Austin.

Send correspondence to Sungho Kang, Semiconductor Systems Design Technology, Motorola Inc., Mail Drop OE321, 6501 William Cannon Drive West, Austin, TX 78735-8598; kang@ssdt-oakhill.sps.mot. com.



Stephen A. Szygenda is a professor and department chair of the Electrical and Computer Engineering Department at the University of Texas at Austin. He holds the Clint Murchison, Sr., Chair of Free Enterprise. His past work includes Bell Telephone Laboratories; faculty membership at the University of Missouri, SMU, and the University of Texas; founding of CCSS, a multiproduct simulation and test company; and founding of the Rubicon Group, a high-technology incubator. He received his MS and PhD degrees from Northwestern University and has been active in numerous areas of the IEEE.