LOGIC DESIGN VALIDATION VIA SIMULATION AND AUTOMATIC TEST PATTERN GENERATION[1]

Hussain Al-Asaad* and John P. Hayes**

*Computer Engineering Research Laboratory
Department of Electrical and Computer Engineering
University of California
One Shields Avenue, Davis, CA 95616-5294

**Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
1301 Beal Avenue, Ann Arbor, MI 48109–2122

Submitted August 1999, Revised April 2000

ABSTRACT

We investigate an automated design validation scheme for gate-level combinational and sequential circuits that borrows methods from simulation and test generation for physical faults, and verifies a circuit with respect to a modeled set of design errors. The error models used in prior research are examined and reduced to five types: gate substitution errors (GSEs), gate count errors (GCEs), input count errors (ICEs), wrong input errors (WIEs), and latch count errors (LCEs). Conditions are derived for a gate to be testable for GSEs, which lead to small, complete test sets for GSEs; near-minimal test sets are also derived for GCEs. We analyze undetectability in design errors and relate it to single stuck-line (SSL) redundancy. We show how to map all the foregoing error types into SSL faults, and describe an extensive set of experiments to evaluate the proposed method. These experiments demonstrate that high coverage of the modeled errors can be achieved with small test sets obtained with standard test generation and simulation tools for physical faults.

Index Terms: Design validation, error modeling, fault simulation, logic design, test generation.

---

[1] A preliminary version of this paper was presented in [6]. The research was supported in part by the National Science Foundation under Grant No. CCR–9872066, by General Motors R&D Center, by DARPA under Contract No. DABT63-96-C-0074, and by start-up funds from the Department of Electrical and Computer Engineering at the University of California—Davis. The results presented herein do not necessarily reflect the position or the policy of the U.S. Government.

1 INTRODUCTION

Design verification is the process of ensuring that a design exhibits the required behavior. The two main approaches to verifying logic designs are formal verification [13][24][28][34][35], which attempts to prove mathematically that a design's implementation and specification agree, and simulation [2][10][14][20][21], which requires the generation and application of input patterns (tests) that check for correct behavior. Simulation is the most widely used and practical verification technique for large logic circuits. Exhaustive simulation using all possible input combinations as tests is one possibility, at least for combinational logic, but the number of tests needed for adequate coverage of design errors tends to be excessive (exponential). An alternative is to perform simulation with randomly generated test vectors [21]. Random vectors can cover a substantial number of design faults, but their coverage is uncertain even with very large test sets. Implementation-independent "universal" test sets [10][14] have also been proposed for design error detection and, more recently, for block-level fault detection [22]. Universal tests exploit any unateness properties of the functions being implemented, but the tests become exhaustive when, as is often the case, there are no unate variables. Yet another approach is to use specific, deterministic test sets generated for a physical (fabrication) fault model like the single stuck-line (SSL) model to verify the design. It has been shown that many gate-level design errors can be detected by using test sets derived for SSL faults [2].

It is also possible to simulate design errors explicitly and generate tests for the resulting models. This nonexhaustive and deterministic approach underlies the mutation technique for software testing [23], and has also been applied to hardware design verification [2][14][21]. This paper investigates model-based design verification for both combinational and sequential logic circuits. Since no criterion for the completeness of a set of design error models is known, a design that passes the tests is guaranteed to be correct with respect to the modeled faults only. In spite of this limitation, model-based simulation appears to be an effective technique for hardware design verification, and can help to discover design faults early in the design process.

We have been exploring model-based design verification at several levels of abstraction [4][5]. Our experience to date suggests that it is difficult to identify generally-applicable design error models at the register-transfer (RTL) or higher levels, but that it is quite feasible to do so at the gate level as this paper shows. Furthermore, gate-level error models can provide useful clues to the nature and capabilities of higher-level models.

The method presented in this paper complements rather than replaces existing approaches for formal gate-level design verification such as logic/property checking [29]. It has several advantages over these approaches. For example, it can validate designs such as multipliers that have outputs requiring very large binary decision diagrams. It can also validate sequential gate-level implementations against high-level specifications without the need to have an explicit mapping of storage elements from the specification to the implementation. The generated deterministic test sets also appear to be useful for error location, diagnosis, correction, and detection of manufacturing faults and can readily serve as input to existing algorithms for design error diagnosis [9][18]. This is the case since these test sets (simulation vectors) can be surprisingly small and can guarantee the detection of broad categories of design errors. In contrast, random vectors [18][30] do not guarantee the detection of all errors, and the use of exhaustive tests [30] is rarely feasible.

In Section 2, we examine the previously proposed design error models [1][2][14][21], and reduce them to five classes. Then we study in detail the detection requirements of these error classes. Section 3 describes the mapping of design errors into SSL faults, as well as the process of generating tests for them using standard test generation and simulation tools for SSL faults. Section 4 presents the results of applying our method to representative combinational and sequential benchmark circuits. Section 5 briefly discusses the extension of the method to RTL circuits.

2 TESTS FOR DESIGN ERRORS

Many types of design errors affecting logic circuits are identified in the research literature [1][2][14][21]. These error types are not necessarily complete, but they are believed to be common in both manual and automated logic synthesis. We condense the errors identified by Abadir et al. [2] into four categories. (A similar classification is given independently in [14]). We also add a fifth category for sequential circuits.

- *Gate substitution error (GSE):* This refers to mistakenly replacing a gate by another gate with the same number of inputs. The extra and missing inverter errors of [1][2][14][21] are considered as substitution of an inverter for a buffer, and a buffer for an inverter, respectively.
- *Gate count error (GCE):* This corresponds to incorrectly adding or removing a gate, and includes the extra and missing gate errors of [2]. This category is combined with gate substitution in [14], where, unlike here, XOR and XNOR gates are not considered. A class of “local”

errors is defined in [21] which includes only some of the errors in this category.

- *Input count error (ICE):* This corresponds to using a gate with more or fewer inputs than required.
- *Wrong input error (WIE):* This error corresponds to connecting a gate input to a wrong signal. The "signal-like-source" error [21], is a special case of WIE. Although a WIE may be viewed as a multiple ICE, a multiple ICE cannot model a WIE in an inverter.

We further identify the following error model for sequential circuits:

- *Latch count error* (*LCE*): This error occurs when a latch is incorrectly added or omitted, due to human error or using imperfect CAD tools for synthesis or (re) timing analysis.

The errors in each category are studied next, and tests needed to detect them are determined. The following assumptions are made concerning the design to be verified:

- A gate-level implementation is available that is either combinational or synchronous sequential.
- The gate types used are AND, OR, XOR, NAND, NOR, XNOR, BUF (buffer) and NOT.
- As in [2][14][21], a functional specification of the design is available which is completely simulatable, that is, any input pattern (sequence) can be applied and produces a completely specified output pattern (sequence).
- At most one design error is present. This assumption is made in the standard SSL model and, indeed, in most other models used in testing for physical faults.

2.1 NOTATION

Let E be the set of all 2^n input vectors of an n-input gate G. We divide E into the disjoint subsets V_0, V_1,..., V_n, where V_k contains all input vectors with exactly k 1s in their binary representation, $0 \le k \le n$. Particularly useful are the disjoint sets V_{null}, V_{all}, V_{odd}, and V_{even} defined as follows:

$$V_{null} = V_0 \qquad V_{all} = V_n$$

$$V_{odd} = \bigcup_{i = odd \wedge i \neq n} V_i \qquad V_{even} = \bigcup_{i = even \wedge i \neq 0 \wedge i \neq n} V_i$$

For example, in the case of 3-input NAND gate, $V_{null} = \{000\}$, $V_{all} = \{111\}$, $V_{odd} = \{001,010,100\}$, and $V_{even} = \{011, 101, 110\}$. We call V_{null}, V_{all}, V_{odd}, and V_{even} the *characterizing sets* or *C-sets* of G.

Table 1 shows the output responses of each gate type to its various C-sets. The sets V_{null} and

Table 1 Responses of the various gate types to their C-sets.

C-set	***n* = 1**		***n* even (*n* odd and $n \geq 3$)**					
	NOT	**BUF**	**AND**	**NAND**	**OR**	**NOR**	**XOR**	**XNOR**
V_{null}	1	0	0 (0)	1 (1)	0 (0)	1 (1)	0 (0)	1 (1)
V_{even}	n/a	n/a	0 (0)	1 (1)	1 (1)	0 (0)	0 (0)	1 (1)
V_{odd}	n/a	n/a	0 (0)	1 (1)	1 (1)	0 (0)	1 (1)	0 (0)
V_{all}	0	1	1 (1)	0 (0)	1 (1)	0 (0)	0 (1)	1 (0)

V_{all} are nonempty and always have cardinality one. For the single-input gates, V_{even} and V_{odd} are empty. For multiple-input gates, the set V_{odd} contains at least two elements, while the set V_{even} is empty only when $n = 2$. The cardinality of V_{even} (V_{odd}) is $2^{n-1} - 1$ ($2^{n-1} - 1$) when n is odd, and $2^{n-1} - 2$ (2^{n-1}) when n is even. Finally, v_k denotes an arbitrary vector of the set V_k.

The above notation enables us to express sets of vectors in a concise way. For example, the complete test set for SSL faults in an n-input NAND gate is $V_n \cup V_{n-1}$. When $n = 3$, we can also write these tests as $V_{all} \cup V_{even} = \{111, 011, 101, 110\}$. In general, to verify the identity of a gate *G*, that is, to determine the tests required for its verification, we use the above notation in conjunction with Table 1.

2.2 GATE SUBSTITUTION ERRORS (GSES)

According to experiments reported in [1], the most frequent error made by human designers is gate substitution, accounting for around 67% of all errors. Gate substitution refers to mistakenly replacing a gate (in a schematic) or a gate operator (in an HDL description) *G* with another gate *G'* that has the same number of inputs. We represent this error by G/G'. For gates with multiple inputs, a *multiple-input GSE* (*MIGSE*) can have one of six possible forms: G/AND, G/NAND, G/OR, G/NOR, G/XOR, and G/XNOR. Each multiple-input gate can have five MIGSEs. For example, all MIGSEs can occur on an AND gate except G/AND which is not considered an error. For gates with a single input, i.e., buffers and inverters, a *single-input GSE* (*SIGSE*) can have one of two possible forms: G/NOT and G/BUF. Each single-input gate can have only one SIGSE. To cover extra or missing inverters in GSEs, a buffer can be inserted in each of a gate's fanout branches as well as in inputs that fan out.

It has been suggested that most GSEs can be detected by a complete test set for SSL faults [2]. Our simulation study (Section 4) shows that such a test set can cover 80% to 100% of MIGSEs and 100% of SIGSEs. The actual coverage of MIGSEs is a function of the circuit structure, as well as the types of gates used in the circuit. Our goal here is to achieve 100% coverage for GSEs.

A single-input gate can be identified by one test vector from either V_{null} or V_{all}. On the other hand, a multiple-input gate can be identified by three test vectors: one from V_{null}, one from V_{odd}, and one from V_{all} (if n is even) or V_{even} (if n is odd). Hence, three test vectors are sufficient to identify an n-input gate. Two test vectors suffice in some cases. For example, an AND gate can be identified by applying one test vector from V_{null} and one from V_{odd}.

The number of tests needed to test an n-input gate for SSL faults is $n + 1$ for the gates AND, NAND, OR, and NOR, while it is two or three for XOR and XNOR depending on the parity of n. So, the number of tests needed to test for SSL faults is greater or equal to the number of tests needed to test for MIGSEs in most cases.

We now introduce some notation to specify the effects of C-sets on a gate G within a circuit.

Definition 1 If the inputs of a gate G in a circuit C can be forced to the pattern v by assigning the primary inputs of C, then G is *controllable* by v; otherwise, it is *uncontrollable* by v. If the output of G with respect to the pattern v is sensitizable to a primary output then the response of v is said to be *observable* at G; otherwise, it is *unobservable*.

Definition 2 A gate G in a circuit C is *V-controllable* if G is controllable by at least one vector v in the input vector set V. If v is also observable at G, then G is *excitable* by V (*V-excitable*). A gate G is *fully excitable* if G is excitable by every nonempty C-set of G; otherwise, it is *partially excitable*.

To illustrate these definitions, consider the circuit in Figure 1. G_1 and G_2 are both controllable by the pattern 00, while G_3 is uncontrollable by 00. The response to the pattern 00 is observable at G_2 but it is not observable at G_1. The gate G_3 is $\{00,11\}$-excitable because G_3 is controllable by 11, and the response of 11 is observable at G_3. However, G_3 is not $\{00\}$-excitable because G_3 is uncontrollable by all the elements of the set $\{00\}$. The gates G_1, G_2, and G_3 are partially excitable.

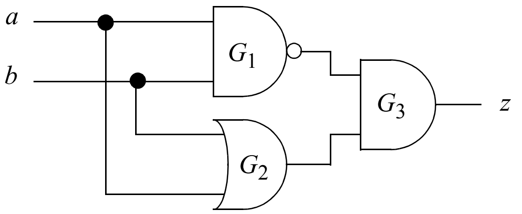

Figure 1 Circuit realizing the XOR function.

The following theorem solves the verification problem for GSEs:

Theorem 1 *A necessary and sufficient condition for a test set S to verify a fully excitable gate is that S produce the test vectors* T_1 *shown in Table 2 at the inputs of the gate and sensitize the gate output to a primary output.*

Proof: We prove the case for an AND gate with an odd number of inputs only; the other cases can be proved similarly. Sufficiency follows directly from Table 2. To prove necessity, assume that S verifies the AND gate G but does not produce either $\{v_{all}, v_{odd}\}$ or $\{v_{null}, v_{odd}\}$ at G's inputs. It is clear from Table 2 that there is no single vector capable of verifying an AND gate. Hence, the set S produces one of the following sets at the inputs of G: $\{v_{null}, v_{all}\}$, $\{v_{null}, v_{even}\}$, $\{v_{even}, v_{all}\}$, $\{v_{odd}, v_{even}\}$, or $\{v_{null}, v_{even}, v_{all}\}$. Table 1 shows that none of these are capable of verifying the AND gate. Hence S must produce $\{v_{all}, v_{odd}\}$ or $\{v_{null}, v_{odd}\}$ at G's inputs. Note that the above analysis implies that two test vectors are sufficient to verify a fully excitable gate. □

All gates in a fanout-free circuit are fully excitable. In a circuit with fanout, it is possible that some input combinations cannot be forced at the inputs of some gates. For example, no element of V_{null} can be forced at the inputs of the AND gate G_3 in Figure 1. From Table 1 we see that V_{null} is necessary to distinguish a 2-input AND gate from an XNOR gate, so, the replacement of the AND by an XNOR gate cannot be detected. This replacement does not change the function of the circuit, hence it is considered to be an *undetectable* MIGSE. Likewise, some input combinations can

Table 2 The test vectors required to verify an *n*-input gate.

Gate	Fanin *n*	Test set T_1	Test set T_2	Test set T_3
NOT (BUF)	$n = 1$	$\{v_{all}\}$ or $\{v_{null}\}$	$\{v_{all}, v_{null}\}$	$\{v_{all}, v_{null}\}$
AND (NAND)	$n = 2$	$\{v_{null}, v_{odd}\}$	$\{v_{all}, v_{odd}, v_{null}\}$	$\{v_{all}, v_{null}, v_{odd}\}$
	n odd	$\{v_{all}, v_{odd}\}$ or $\{v_{null}, v_{odd}\}$	$\{v_{all}, v_{odd}, v_{null}\}$	$\{v_{null}, v_{odd}, v_{all}, v_{even}\}$
	n even & $n \neq 2$	$\{v_{null}, v_{odd}\}$ or $\{v_{all}, v_{even}\}$	$\{v_{null}, v_{odd}, v_{all}, v_{even}\}$	$\{v_{null}, v_{odd}, v_{all}, v_{even}\}$
OR (NOR)	$n = 2$	$\{v_{all}, v_{odd}\}$	$\{v_{all}, v_{odd}, v_{null}\}$	$\{v_{all}, v_{null}, v_{odd}\}$
	n odd	$\{v_{null}, v_{even}\}$ or $\{v_{all}, v_{even}\}$	$\{v_{null}, v_{even}, v_{all}\}$	$\{v_{null}, v_{odd}, v_{all}, v_{even}\}$
	n even & $n \neq 2$	$\{v_{null}, v_{even}\}$ or $\{v_{all}, v_{odd}\}$	$\{v_{null}, v_{even}, v_{all}, v_{odd}\}$	$\{v_{null}, v_{odd}, v_{all}, v_{even}\}$
XOR (XNOR)	$n = 2$	$\{v_{null}, v_{all}\}$	$\{v_{all}, v_{odd}, v_{null}\}$	$\{v_{all}, v_{null}, v_{odd}\}$
	n odd	$\{v_{odd}, v_{even}\}$	$\{v_{odd}, v_{even}, v_{all}\}$ or $\{v_{odd}, v_{even}, v_{null}\}$	$\{v_{null}, v_{odd}, v_{all}, v_{even}\}$
	n even & $n \neq 2$	$\{v_{null}, v_{all}\}$ or $\{v_{even}, v_{odd}\}$	$\{v_{null}, v_{all}, v_{even}, v_{odd}\}$	$\{v_{null}, v_{odd}, v_{all}, v_{even}\}$

be forced at the inputs of some gates but their responses cannot be observed. For example, the pattern 00 can be forced at the inputs of G_1 in Figure 1, but the response of G_1 cannot be propagated to the primary output. The above examples show that it is natural to have gates which are not fully excitable and therefore have undetectable design errors. It also suggests a modification of the test vectors T_1 in Table 2 to verify a partially excitable gate.

Definition 3 If a partially excitable gate G is excitable by all but one of its nonempty C-sets, then G is *strong partially excitable*, otherwise, it is *weak partially excitable*.

Consider again the circuit in Figure 1. The gates G_1, G_2, and G_3 are strong partially excitable because they are excitable by two out of the three nonempty C-sets of the respective gates. An example of a weak partially excitable gate is a 3-input XOR with all inputs connected to a single source. In this case, the gate is excitable by only two (V_{null} and V_{all}) of its four C-sets.

Since a strong partially excitable gate G is not excitable by one of the nonempty C-sets, one of its MIGSEs is undetectable. The remaining four MIGSEs on G can be detected with at least two vectors; Table 1 implies that an arbitrary vector detects only three of G's five MIGSEs. Therefore, we have to apply at least three test vectors to G, so that if G is not controllable by one of the vectors or one of the vectors' responses is not observable, then the other two will detect the detectable MIGSEs. This leads to the following result.

Theorem 2 *If all gates of a circuit are either fully excitable or strong partially excitable, then the test set T_2 shown in Table 2 detects all detectable GSEs in the circuit.*

Proof: If a gate G is fully excitable, then G is controllable by the test vectors in T_2 and their responses are observable at G. Since each test set in T_2 for a particular gate is a superset of the test set in T_1 for the same gate, all GSEs will be detected. If, on the other hand, G is strong partially excitable, then it is not controllable by one of the test vectors in T_2, or the response of one of the vectors is not observable at G. It follows from Table 1 that if we remove a test vector for G from T_2, then the remaining vectors detect all the detectable GSEs on that gate. □

A further analysis of T_2 shows that to verify a weak partially excitable gate, we have to apply the patterns T_3 shown in Table 2. Since we cannot always assert that the gates in the design under test are fully excitable or strong partially excitable, we may have to apply the patterns T_3 to detect all GSEs. Note that a test set generated for GSEs assuming that the gates are weak partially excitable, will detect all GSEs in the circuit. On the other hand, a test set generated for GSEs by assuming the gates are fully excitable or strong partially excitable may not detect all GSEs.

A complete test set for SSL faults guarantees the detection of all SIGSEs [2]. Tests for MIGSEs also cover many SIGSEs. A circuit is *SSL-irredundant* if it has no undetectable SSL faults.

Theorem 3 *A complete test set T for MIGSEs in an SSL-irredundant circuit is also a complete test set for SIGSEs on all circuit lines except inputs with fanout, if T produces* v_{all} *at the input of every AND and NAND gate,* v_{null} *at the input of every OR and NOR gate, and their responses are observable.*

From this theorem we conclude that detection of most SIGSEs is ensured by test sets T_2 and T_3 but not by T_1. Our experiments show that the test set T_3 detects all SIGSEs in all SSL-irredundant benchmark circuits considered in Section 4.

2.3 GATE COUNT ERRORS (GCES)

We distinguish two types of gate count errors: extra-gate and missing-gate errors. An *extra-gate design error* (EGE) is defined as inserting a gate *G'* that has its *m* inputs taken from the *n* inputs of a gate *G* and feeding the output of *G'* to *G*. As a consequence, the number of inputs of gate *G* becomes $n - m + 1$. We represent an EGE by EG(*G'*,*G*). It is easily seen that EG(AND,AND), EG(AND,NAND), EG(OR,OR), EG(OR,NOR), EG(XOR,XOR), and EG(XOR, XNOR) are undetectable. Explicit test generation for EGEs is not needed due to the following result.

Theorem 4 *A complete test set for GSEs is also a complete test set for EGEs.*

Proof: An EGE can be mapped easily into a GSE. EG(*G'*,*G*) is nothing but the GSE *G"*/*G'*, where *G"* is determined by *G* as follows: (1) if *G* is an AND or NAND, then *G"* is an AND; (2) if *G* is an OR or NOR, then *G"* is an OR; (3) if *G* is an XOR or XNOR, then *G"* is an XOR. Hence, any test set that detect all GSEs will detect all EGEs. □

Most EGEs can also be detected by a complete test set for SSL faults, but this is not guaranteed. A complete test set for SSL faults in the circuit of Figure 2 is {000, 100, 001, 010}. This test

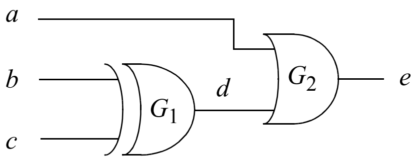

Figure 2 Example showing an EGE that is not detected by a complete test set for SSL faults.

set does not detect if the XOR gate is an extra gate. For that, we need the test 011.

A *missing-gate design error* (MGE) is defined as removing a gate *G'* that has *m* inputs and feeds an *n*-input gate *G*, and then changing the inputs of *G'* into inputs of *G*; see Figure 3. As a consequence, the number of inputs of *G* becomes $N = n + m - 1$. We represent the MGE by MG(*G'*,*G*). As in the extra-gate case, the errors MG(AND, AND), MG(AND, NAND), MG(OR, OR), MG(OR, NOR), MG(XOR,XOR), and MG(XOR,XNOR) are undetectable.

Consider the problem of finding a minimal set of vectors that detect all MGEs in an *N*-input gate *G*. For each MG(*G'*,*G*), we insert a gate *G"* as shown in Figure 4, where *G"* is chosen so that the function of the circuit is not changed. For example, if *G* is an AND or NAND, then *G"* is an AND gate. We have to detect the GSE G"/G' in order to detect MG(*G'*,*G*).

Theorem 5 *The test sets* $V_N \cup V_{N-1} \cup V_{N-2}$, $V_0 \cup V_1 \cup V_2$, *and* $V_0 \cup V_2 \cup V_N$ *are each sufficient and near-minimal for detecting MGEs on an N-input fully excitable AND (or NAND), OR (or NOR), and XOR (or XNOR) respectively.*

Each test set defined by this theorem, which is proven in [4], has one test more than the minimum. For example, the 11-member test set generated for MGEs in a 4-input NAND gate *G* is *S* = {1111, 1110, 1101, 1011, 0111, 1100, 1010, 1001, 0110, 0101, 0011}. If one of the tests {1110, 1101, 1011, 0111} is dropped, *S* still detects all MGEs. However, all MGEs in *G* cannot be detected with fewer than 10 vectors. In general, Theorem 5 gives near-minimal test sets for an *N*-input fully excitable gate. It is easy to prove that these test sets detect all the MGEs of an *N*-input partially excitable gate with high probability.

Figure 3 The missing-gate design error (MGE).

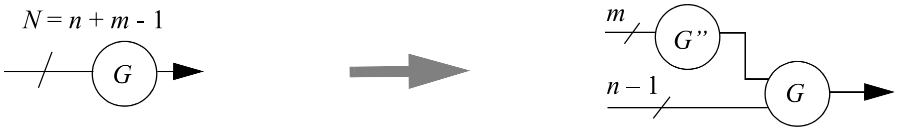

Figure 4 Reducing the problem of detecting MGEs to detecting GSEs.

2.4 INPUT COUNT ERRORS (ICES) AND WRONG INPUT ERRORS (WIES)

Input count errors (ICEs) are classified into extra input and missing input errors. An *extra input design error* (EIE) is defined as the replacement of an *n*-input gate ($n \geq 2$) by an ($n + 1$)-input gate with the additional input connected to an arbitrary signal in the circuit. A *missing input design error* (MIE) is the replacement of a gate of ($n \geq 3$) inputs by an ($n - 1$)-input gate whose $n - 1$ inputs are connected to an arbitrary subset of the original *n*. We represent an EIE of a gate *G* by EI(*e*,*G*) where *e* is the extra input. We represent an MIE of a gate *G* by MI(*m*,*G*) where *m* is the source of the missing input.

To detect an EIE at a given input of an AND or NAND gate, that input must be set to 0 to activate the error, the other inputs must be forced to 1, and the gate's output signal must be propagated to a primary output. This is exactly the requirement of a test for a stuck-at-1 fault at the input of the gate in question. Similarly, testing for EIEs at input *x* of an OR or NOR gate is the same as testing for *x* stuck-at-0. To test for an MIE on an AND gate *G*, the inputs of *G* are set to 1, the signal considered to be missing is set to 0, and *G*'s output signal is propagated to a primary output. This is more restrictive than a test for stuck-at-0 at the output of *G*. Similarly, testing for an MIE on a NAND, OR, and NOR is more restrictive than testing the gate output for stuck-at-1, stuck-at-1, and stuck-at-0, respectively.

The foregoing tests are complete for AND, NAND, OR, and NOR gates. Hence, a complete test set for ICEs in a given circuit detects all SSL faults at AND, NAND, OR, and NOR gates. A complete test set for ICEs also detects some SSL faults affecting XOR and XNOR gates. For example, testing for EIEs at the input of an XOR or XNOR gate is equivalent to testing for a stuck-at-0 fault at the same input.

A *wrong input error* (WIE) is defined as a connection of a gate input to a wrong signal source. We represent a WIE on a gate *G* by WI(*u*,*w*,*G*), where *u* is the wrong input of the gate and *w* is the correct input. If a test vector *v* detects WI(*u*,*w*,*G*), then it must set *u* and *w* to opposite values and propagate the signal at *u* to a primary output. WIE appears to be the second most common design error—around 17% of the errors reported in [1]. The relationship between MIEs and WIEs is stated in the following theorem:

Theorem 6 *A complete test set for MIEs on gates of type AND, NAND, OR, or NOR is a complete test set for WIEs on the same gates.*

Proof: Let G be an AND gate with inputs $x_1, x_2, ..., x_n$ and output y. Consider the WIE WI(x_i,z,G), where z is an arbitrary signal in the circuit. The complete test set for MIEs on G will detect MI(z,G) and hence set the inputs of the gate to 1s, propagate y to a primary output, and set z to 0 with at least one vector v of the test set. Since v sets x_i and z to opposite values, and propagates x_i to a primary output, WI(x_i,z,G) is detected for every i. A similar argument holds for the other gate types. □

In practice, it is hard to find a complete test set for MIEs. The fact that a given MI(x,G) is undetectable does not imply that the WI(u,x,G) is undetectable for every u. Also, a complete test set for MIEs does not guarantee the detection of WIEs in XOR, XNOR, NOT, and BUF gates. Hence, we cannot conclude that a test set for MIEs covers all WIEs.

The numbers of ICEs and WIEs in a circuit are large—approximately $O(k^2)$, where k is the number of distinct signals in the circuit. Hence, we use simulation to extract the errors detected by the test set $S_T = S_{SSL} \cup S_{GSE} \cup S_{MGE}$, where S_{SSL}, S_{GSE}, and S_{MGE} are complete test sets for SSL faults, GSEs, and MGEs, respectively. In fact, all EIEs are detected by the test set for SSL faults alone [2], hence, we only have to generate tests for the undetected MIEs and WIEs. Our experimental results show that most MIEs and WIEs are detected by the set S_T.

A basic question concerning MIEs (WIEs) is the source of the missing (wrong) input. It must not depend on the erroneous gate's output, otherwise, the circuit can become sequential and asynchronous. Errors that make a circuit asynchronous can be detected by a levelization procedure [3].

The coverage relationships among the various design errors are summarized as follows. A complete test set for MIGSEs detects all EGEs. On the other hand, a complete test set for SSL faults detects all EIEs and SIGSEs. Complete test sets for MIEs, MGEs, and WIEs do not guarantee the detection of other error types. For example, a test for MIEs detects many, but not necessarily all, SSL faults.

2.5 LATCH COUNT ERRORS (LCES)

Latch count errors (LCEs) are classified into extra and missing latch errors. We assume that all latches are of the clocked D type, synchronized by a common clock signal. An *extra latch design error* (*ELE*) is defined as the insertion of a latch into any line in the circuit. A *missing latch design error* (*MLE*) is the replacement of a latch by a line. It is impractical to consider all possible MLEs due to their impact on the circuit's state space and test generation complexity. Hence, we only consider MLEs affecting the circuit's primary inputs and outputs.

In contrast to the design errors studied in the previous subsections, to check for LCEs, a test sequence rather than an unordered set of test patterns is needed. To test for an ELE, a transition sequence, either $0 \rightarrow 1$ or $1 \rightarrow 0$, is applied at the input of the latch and its response is propagated to a primary output. Similarly, to test for an MLE, a transition sequence is applied at the line where the latch may be missing and the transition sequence is propagated to a primary output.

2.6 DESIGN ERROR UNDETECTABILITY

We noted earlier that some design errors are undetectable. This leads to a type of redundancy that is quite different from that previously studied [17].

Definition 4 A gate *G* in a circuit *C* has *redundant inputs* if the function implemented by *C* is not changed when a proper subset of the inputs of *G* are removed. A circuit *C* is called *GI-irredundant* if no gate in *C* has redundant inputs.

GI-redundancy does not imply SSL-redundancy. For example, a 5-input XOR with all inputs connected to the same source is GI-redundant but SSL-irredundant. Similarly, SSL-redundancy does not imply GI-redundancy. For example, a buffer whose input is connected to ground is SSL-redundant but GI-irredundant.

An *undetectable design error* is one for which no test vector exists. For example, the substitution of an XNOR gate for G_3 in Figure 1 cannot be detected by any input vector. Hence, the MIGSE G_3/XNOR is undetectable. The following theorem [4] characterizes undetectable GSEs:

Theorem 7 *In a GI-irredundant and SSL-irredundant circuit C, the following holds: (1) C has no undetectable SIGSEs; (2) If G/G' is an undetectable MIGSE then every other MIGSE on G is detectable, and if* $G \in \{XOR, XNOR\}$ *then* $G' \in \{AND, NAND, OR, NOR\}$ *and vice versa.*

Corollary 1 *If the gates in a GI-irredundant and SSL-irredundant circuit C are restricted to AND, NAND, OR, NOR, NOT, and BUF, then C has no undetectable GSEs.*

The number of gates that can have undetectable MIGSEs in a circuit *C* varies with the circuit structure and the types of gates in *C*. For example, fanout-free circuits have no undetectable GSEs. On the other hand, a 2-input XOR circuit implemented using four 2-input NAND gates has four possible undetectable MIGSEs: each NAND gate can be replaced with an XOR without affecting the overall XOR function.

It is interesting to observe that an undetectable design error can be interpreted as a permissible transformation of the circuit [15][25][31]. In fact, logic design verification may be seen as the inverse of the network transformation performed during logic synthesis. While the former tries to detect all non permissible transformations (design errors), the latter attempts to find all permissible transformations (feasible designs).

3 VERIFICATION TEST GENERATION

This section describes our method for modeling and detecting design errors. In order to use standard ATPG tools, we map the error types under consideration into SSL faults. The mapping process consists of modifying the target circuit's netlist (or equivalent description) and injecting a predefined set of SSL faults. A test set is then generated for these faults in the modified netlist which detects all errors in the original design.

To map MIGSEs and MGEs into SSL faults, each gate in the original netlist is replaced by a functionally equivalent circuit called a *gate replacement module*. A few selected SSL faults are injected in the gate replacement module, so that the test for each injected fault forces the input of the gate to be a vector from one of the sets required to verify the gate. To cover all possible MIGSEs in a circuit, we must assume that the gates are weak partially excitable. Consider, for example, the AND replacement module shown in Figure 5. The faults *c* stuck-at-0, *d* stuck-at-0, and *e* stuck-at-0 will force the inputs of the AND replacement module to v_{null}, v_{odd}, and v_{all}, respectively. These input patterns determine if the AND gate in the circuit is correct or not, i.e., the presence of any MIGSE on the gate is detected. The gate replacement modules for MIGSEs and MGEs on all gate types can be designed systematically in a similar fashion [4].

The requirements to be met by a gate replacement module *M(G)* of a gate *G* are as follows:

- The function of *M(G)* must be the same as that of *G*.
- A test for an injected SSL fault in *M(G)* must force the input of *G* to a certain vector that is

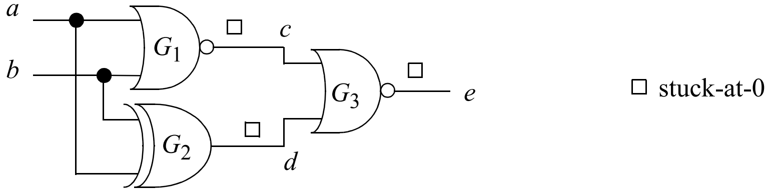

Figure 5 The replacement module for detecting GSEs in a 2-input AND gate.

needed to verify *G*.

- The injected SSL faults must be sensitizable to the output of *M(G)*.
- If an injected SSL fault in M(G) is detected by a vector $v \in V_i$, then it must be detected by any vector of V_i. This requirement simplifies the detection of the injected SSL faults by the test generator, and leads to smaller test sets.

The mapping of MIGSEs and MGEs into SSL faults is many-to-one. Detecting a given set of injected SSL faults detects a larger set of MIGSEs and MGEs. For example, detection of the three SSL faults in Figure 5 detects five MIGSEs. There is a one-to-one correspondence between net errors (EIEs, MIEs, and WIEs) and SSL faults. The mapping of an EIE into an SSL fault is very simple: to detect whether an AND or NAND gate's input *x* is extra, we need to set *x* to 0, set every other input to 1, and propagate the gate's output signal to a primary output. This is the same as testing for *x* stuck-at-1. Also, to test for an extra input in an OR, NOR, XOR, or XNOR gate, a test for the input stuck-at-1 is required.

The detection of MIEs and WIEs is modeled by a mapping circuit called a *net attachment module*, as shown in Figure 6. Let *C* and *C'* be the circuits obtained before and after adding the net attachment module. The following requirements must be met:

- The function of circuit *C* must be the same as that of *C'*.
- A test for the injected SSL fault in the net attachment module must detect the MIE or WIE.
- The injected SSL fault must be sensitizable in the net attachment module.

A typical design of a net attachment module for MIEs appears in Figure 7a. If G_2 is an AND or NAND, then G_1 must be an XNOR and the fault *p* stuck-at-1 is injected. On the other hand, if G_2

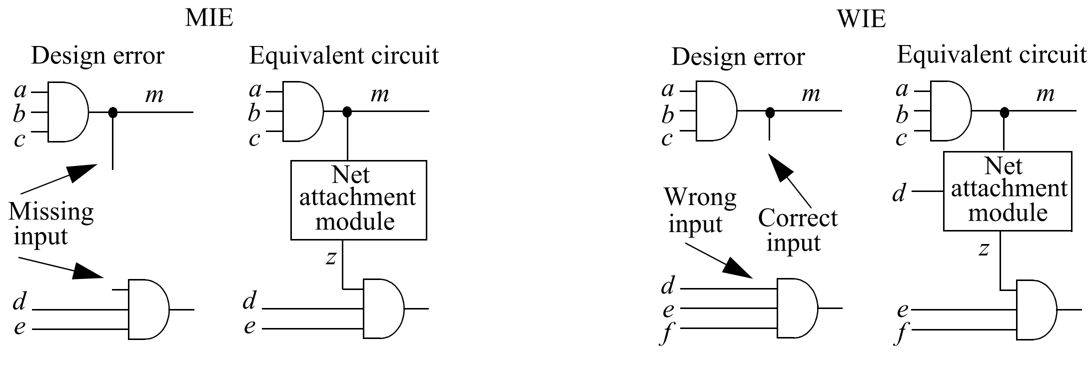

Figure 6 Mapping MIEs and WIEs into SSL faults.

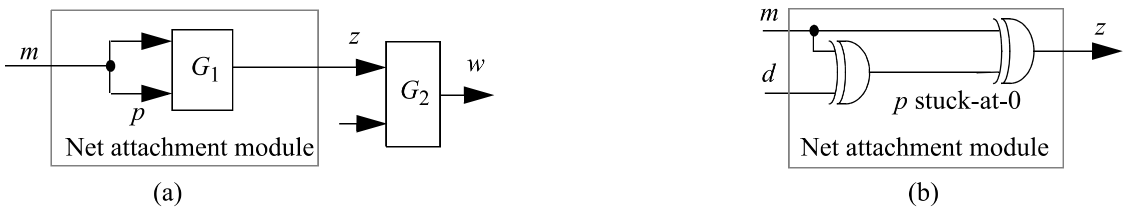

Figure 7 A net attachment module (a) for MIEs and (b) for WIEs.

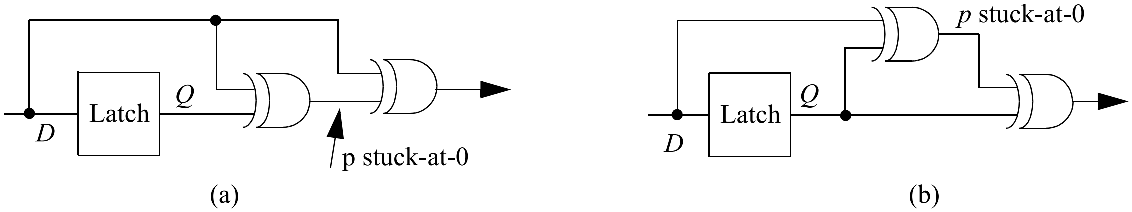

Figure 8 (a) Latch and (b) line replacement modules to detect ELEs and MLEs, respectively.

is any of the gates {OR, NOR, XOR, XNOR}, then G_1 must be an XOR and the fault p stuck-at-0 is injected. In both cases, the output of G_2 is independent of z and hence the function of the circuit is not changed. Also, the SSL fault is sensitizable to the output of the net attachment module and the vector testing it detects MI(m,G_2). A typical design of the net attachment module for a WIE is shown in Figure 7b. The output of the net attachment module is $z = d$, hence the circuit function is preserved. The test for p stuck-at-0 forces opposing values on m and d, and hence the corresponding WIE will be detected by the same test.

The detection of ELEs is performed by replacing the latch by the *latch replacement module* shown in Figure 8(a), and then generating a test sequence for the SSL fault p stuck-at-0, which is also a test sequence for the ELE. Similarly, the detection of MLEs is performed by replacing the line by the *line replacement module* as shown in Figure 8(b). The test sequence generated for the SSL fault p stuck-at-0 is also a test for the MLE. Hence, LCEs are easily mapped into SSL faults.

The overall verification process is divided into two phases. The first phase generate tests for gate errors (MIGSEs and MGEs) and is shown in Figure 9. If the circuit is sequential, additional tests for LCEs are generated. The second phase performs the error simulation for net errors (MIEs, WIEs) and then generates tests for the undetected ones; the flowchart of phase 2 is similar to that of phase 1. Complete coverage of net errors may require several iterations through phase 2. If after checking for all modeled errors, the implementation is found to match the functional specifications, we can conclude with high confidence that the circuit is correct as designed.

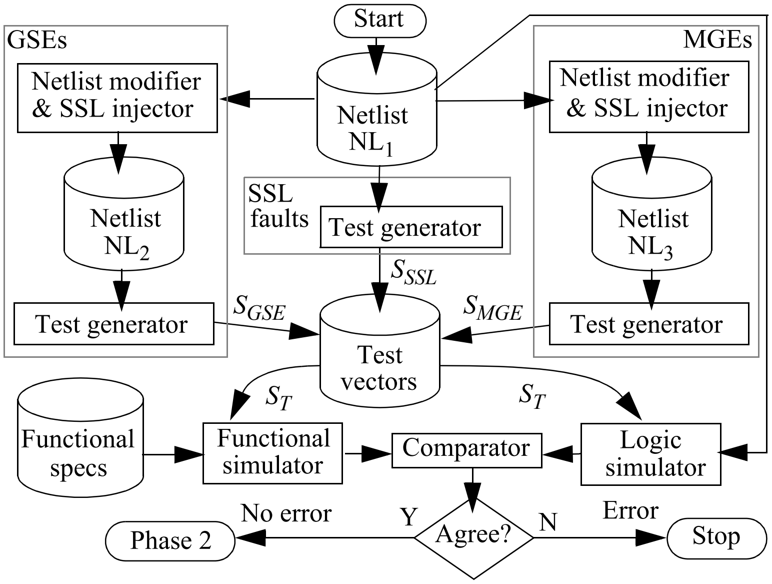

Figure 9 First phase of the design verification process.

4 EXPERIMENTAL RESULTS

In this section we describe the experiments performed to support the preceding analysis; these experiments used the combinational ATPG tool ATALANTA [27] and the sequential ATPG tool ATTEST [8]. To determine the ability of a given test set to detect design errors and SSL faults, we developed an error/fault simulator ESIM [7]. For combinational circuits, the simulator uses parallel-pattern evaluation and critical path tracing [3]; It simulates the circuit with multiple vectors concurrently and determines the detected errors/faults without explicit simulation of each error/fault. ESIM uses parallel fault simulation [3] for sequential circuits.

The circuits used for the experiments are the ISCAS 85 combinational benchmarks [11], some standard, combinational 74X-series circuits [33], and the ISCAS 89 sequential benchmarks [12]. We conducted a preliminary experiment to determine the coverage of design errors using a complete test set for all detectable SSL faults. The resulting data given in Table 3 show that a complete test set for SSL faults guarantees the detection of all SIGSEs and EIEs, confirming results in [2]. The detection of the other design error types is not guaranteed but they are likely to be detected because the test set does exercise each net in the circuit. Note that all the circuits in Table 3 are relatively small and SSL-irredundant.

Our next experiments are concerned with generating nearly complete test sets for all modeled

Table 3 Design error coverage in combinational benchmarks using complete SSL test set generated by ATALANTA.

Circuit	Test set size	Detected SSL faults	Detected GSEs		Detected GCEs		Detected ICEs		Detected WIEs
			SIGSE	MIGSE	EGE	MGE	EIE	MIE	
c17	5	100.0	100.0	80.0	100.0	n/a	100.0	57.5	88.0
c432nr	44	100.0	100.0	89.1	100.0	95.5	100.0	73.1	96.9
c499nr	52	100.0	100.0	97.9	46.2	93.8	100.0	88.8	98.9
c880	47	100.0	100.0	90.3	100.0	94.6	100.0	84.9	98.6
7485	25	100.0	100.0	88.4	100.0	89.8	100.0	83.4	92.7
74181	18	100.0	100.0	96.2	88.9	90.6	100.0	81.8	94.0
74283	12	100.0	100.0	91.3	100.0	84.1	100.0	74.5	92.2

Table 4 Design error coverage in combinational benchmarks using verification tests generated by ATALANTA.

Circuit	Tests targeting GSEs				Tests targeting MGEs		Error simulation for MIEs and WIEs using S_T		
	Test set size	Detected MIGSEs	Detected SIGSEs	Detected EGEs	Test set size	Detected MGEs	Test set size	Detected MIEs	Detected WIEs
c17	5	100.0	100.0	100.0	n/a	n/a	10	82.5	95.7
c432nr	39	92.8	100.0	100.0	92	99.9	174	88.8	99.5
c499nr	39	99.8	100.0	46.2	43	98.4	133	93.2	99.7
c880	49	92.8	100.0	100.0	66	100.0	162	95.0	99.8
7485	14	88.4	100.0	100.0	47	94.4	85	89.3	96.4
74181	15	98.5	100.0	88.9	36	99.5	69	94.9	98.8
74283	10	94.7	100.0	100.0	31	100.0	51	88.7	95.2

design errors. They use the method described in the previous section to generate test vectors targeting specific errors. The modified netlist is supplied to ATALANTA which generates a test set. The generated test sets are then evaluated using simulation to find their coverage of GSEs, as shown in Table 4. Since tests for MIGSEs cover EGEs (Theorem 4), the results on detecting EGEs are shown in Table 4. The coverage of MGEs is also shown in Table 4. Testing for MIEs, and WIEs is performed only for those errors that are not detected by error simulation using the set $S_T = S_{SSL} \cup S_{GSE} \cup S_{MGE}$. The coverage of EIEs is the same as that shown in Table 3 because a complete test set for SSL faults detects all EIEs. The error simulation results for MIEs and WIEs also appear in Table 4. Tests were generated using ATALANTA for the remaining undetected MIEs and WIEs after the error simulation. ATALANTA reported that a large percentage of those errors are undetectable. After adding the generated tests to S_T, the coverage of MIEs and WIEs is

Table 5 Improved coverage of MIEs and WIEs after the second phase of test generation using ATALANTA.

Circuit	Tests targeting MIEs not detected by S_T		Tests targeting WIEs not detected by S_T	
	Total test set size	Detected MIEs	Total test set size	Detected WIEs
c17	13	95.0	12	100.0
c432nr	190	89.9	195	99.6
c499nr	220	95.8	147	99.8
c880	225	96.5	192	99.9
7485	91	91.2	92	96.4
74181	83	96.6	78	98.9
74283	58	90.0	56	96.4

improved, as shown in Table 5.

The coverage of design errors using the generated test sets is quite high, 80%–100% in most cases. We are confident that most detectable design errors of the modeled types are actually detected. To explore this further, we analyzed the 7485, 74181, and 74283 circuits in depth. We found that MIGSEs and EGEs not detected by our test sets are undetectable, and hence, these test sets cover 100% of the detectable MIGSEs and EGEs. We also analyzed the circuit c499nr to determine the reason for the poor coverage of EGEs. It turns out that the ISCAS 85 benchmarks limit the number of inputs of XOR gates to two. Since c499nr has several XOR trees, then most XOR gates are considered redundant gates—they can be combined in one XOR gate with large fanin.

It is difficult to compare the coverage results obtained in this paper to related work in the literature for several reasons: (1) different error models are used; (2) test set sizes are missing from the results of [21]; and (3) standard benchmarks are not used in most prior work. The test generation times for the circuits in Table 4 and Table 5 were found to range from a few seconds to a few minutes on a HALstation 300.

To check that our method can use any standard SSL test generator and to determine the design error coverage for the large SSL-redundant ISCAS 85 circuits, we performed the test generation experiments using the advanced SSL test generator ATTEST [8]. The error simulation results of the generated test sets are shown in Table 6. As expected, the generated test sets are small and have high coverage of the modeled errors.

Table 6 Design error coverage in combinational benchmarks using verification tests generated by ATTEST.

Circuit	Size of S_T	Testing time[a]	Detected SIGSEs	Detected MIGSEs	Detected EGEs	Detected MGEs	Detected EIEs	Detected MIEs	Detected WIEs
c1355	265	3.2	100.0	82.3	100.0	97.1	99.2	83.5	99.3
c1908	465	3.7	100.0	84.8	97.6	90.3	99.2	90.8	97.3
c2670	797	7.9	99.7	87.5	87.6	91.6	93.2	90.4	98.7
c3540	650	11.2	99.3	89.7	90.6	81.6	94.2	88.4	98.6
c5315	1263	8.1	99.8	89.6	98.9	93.8	98.3	99.8	99.5
c6288	324	55.7	99.6	85.8	100.0	n/a	99.3	97.9	99.7
c7552	1364	23.5	100.0	86.6	97.4	91.2	96.4	99.4	98.7

[a] In minutes on a HALstation 300.

Table 7 Design error coverage in sequential benchmarks using verification test sequences generated by ATTEST.

Circuit	Size of S_T	Testing time[a]	Detected SSLs	Detected SIGSEs	Detected MIGSEs	Detected EGEs	Detected MGEs	Detected EIEs	Detected MIEs	Detected WIEs	Detected ELEs	Detected MLEs
s27	49	0	100.0	100.0	95.0	100.0	n/a	100.0	74.7	94.4	100.0	100.0
s208	448	8	95.6	91.3	87.9	87.8	83.8	91.6	72.4	93.0	87.5	81.8
s298	448	27	86.4	91.1	93.6	100.0	96.92	77.5	67.5	84.6	100.0	100.0
s344	264	180	93.9	97.3	90.7	100.0	85.0	91.9	76.9	94.2	100.0	95.0
s349	352	28	94.9	97.7	90.8	100.0	85.0	93.5	79.9	95.9	100.0	95.0
s386	500	132	85.1	97.1	92.7	78.4	98.2	69.6	67.1	87.6	100.0	92.9
s420	499	114	54.5	58.1	69.4	71.9	48.1	51.8	32.8	52.8	81.3	36.8
s641	360	14	87.6	93.5	94.8	73.5	90.6	81.7	65.2	90.4	78.9	89.8

[a] In minutes on a HALstation 300.

We further experimented with the proposed method using a representative set of sequential benchmarks from the ISCAS 89 suite [12]. To simplify test generation, we attached a single clear (reset) input to all storage elements in each circuit. We also used the ATTEST SSL test generator to generate the verification test sequences $S_T = S_{SSL} \cup S_{GSE} \cup S_{MGE} \cup S_{ELE} \cup S_{MLE}$. The simulation results (Table 7) were determined by the error simulator ESIM, and demonstrate the effectiveness of the generated test sequences. The coverage of design errors is high for all circuits, except for s420 whose internal nets have low controllability and observability.

5 DISCUSSION

We have presented an error-based method for verifying logic circuits using standard simulation and ATPG tools. We showed that all common gate-level design errors can readily be mapped into SSL faults, and presented a systematic method to perform this mapping. Our experimental results show that complete test sets for the SSL faults detect almost all detectable design errors. The test sets are small and provide high coverage—the percentage of detected design errors from all modeled errors, detectable and undetectable, is greater than 90% for most of the benchmark circuits. The experiments also show that the fraction of undetectable design errors is significant in practical circuits, even when they are SSL-irredundant. For example, 11.6% of the MIGSEs in the 7485 comparator circuit are undetectable. We ensure full detectability of design errors by injecting SSL faults into a modified netlist and apply an ATPG program to it. Any such program can be used off the shelf, so future improvements in ATPG tools can be applied directly to this type of design error detection.

The verification method considered here can, in principle, be extended to higher levels of abstraction such as the register-transfer or functional level of design. The 74283 carry lookahead adder in Figure 10 is an example of an RTL design, where CLA is a lookahead carry-generation module and the other modules are word gates, all interconnected by multibit buses. The goal is then to validate operation of a circuit with respect to its functional specification [32]. Such functional testing is potentially faster and more efficient than gate-level testing, in that it is independent of implementation details. Possible RTL design error types include: module substitution error (word gate substitution, missing bus inversion, etc.), extra/missing module, and extra/missing input bus. Like gate-level errors, such RTL errors can be mapped into simple RTL fault types such as bus stuck-at 0/1 faults, but suitable high-level ATPG tools for such faults are not yet avail-

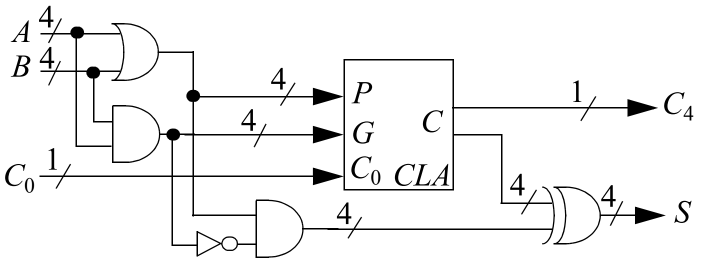

Figure 10 High-level model of the 74283 carry-lookahead adder.

Table 8 Test generation results for RTL design errors in the 74283 adder.

Error type	Test set size	# errors	Detected	Undetectable
Bus stuck-at faults	2	36	36	0
Missing bus inversion	1	18	18	0
Word gate substitution	2	20	19	1
Missing word gate	0	0	0	0
Missing input bus	1	11	9	2
Wrong input bus	2	34	33	1

able. Tests were generated manually for errors of the foregoing types in the 74283 adder, and the results (Table 8) suggest that, as in gate-level design, high coverage of RTL design errors is possible using very few test vectors.

REFERENCES

[1] E. J. Aas, T. Steen, and K. Klingsheim, "Quantifying design quality through design experiments", *IEEE Design and Test*, Vol. 11, pp. 27-37, Spring 1994.

[2] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic design verification via test generation", *IEEE Transactions on CAD*, Vol. 7, pp. 138-148, January 1988.

[3] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, 1990.

[4] H. Al-Asaad, *Lifetime Validation of Digital Systems via Fault Modeling and Test Generation*, Ph.D. dissertation, University of Michigan, 1998.

[5] H. Al-Asaad, D. V. Campenhout, J. P. Hayes, T. Mudge, and R. Brown, "High-level design verification of microprocessors via error modeling", *Digest of Papers: IEEE International High-Level Design Validation and Test Workshop*, 1997, pp. 194-201.

[6] H. Al-Asaad and J. P. Hayes, "Design verification via simulation and automatic test pattern generation", *Proc. International Conference on Computer-Aided Design*, 1995, pp. 174-180.

[7] H. Al-Asaad and J. P. Hayes, "ESIM: A multimodel design error and fault simulator for logic circuits", *Proc. IEEE VLSI Test Symposium*, 2000, pp. 221-228.

[8] *ATTEST Software Tools*, ATTEST Software, Inc., Santa Clara, CA.

[9] V. Boppana et al., "Multiple error diagnosis based on Xlists", *Proc. Design Automation Conference*, 1999, pp. 660-665.

[10] D. Brand, "Exhaustive simulation need not require an exponential number of tests", *IEEE Transactions on CAD*, Vol. 12, pp. 1635-1641, November 1993.

[11] F. Brglez and H. Fujiwara, “A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran”, *Proc. IEEE International Symposium on Circuits and Systems*, 1985, pp. 695-698.

[12] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits”, *Proc. IEEE International Symposium on Circuits and Systems*, 1989, pp. 1929-1934.

[13] R. Bryant, “Graph-based algorithms for boolean function manipulation”, *IEEE Transactions on Computers*, Vol C-35, pp. 677-691, August 1986.

[14] B. Chen, C. L. Lee, and J. E. Chen, “Design verification by using universal test sets”, *Proc. Third Asian Test Symposium*, 1994, pp. 261-266.

[15] S.-C. Chang, M. Marek-Sadowska, and K.-T. Cheng, “Perturb and simplify: multilevel boolean network optimizer”, *IEEE Transactions on CAD*, Vol. 15, pp. 1494-1504, December 1996.

[16] P. Chung, Y. Wang, and I. Hajj, “Logic design error diagnosis and correction”, *IEEE Transactions on VLSI Systems*, Vol. 2, pp. 320-332, September 1994.

[17] J. P. Hayes, “On the properties of irredundant logic networks”, *IEEE Transactions on Computers*, Vol. C-25, pp. 884-892, September 1976.

[18] S.-Y. Huang et al., “ErrorTracer: A fault simulation-based approach to design error diagnosis”, *Proc. International Test Conference*, 1997, pp. 974-981.

[19] J. Jain et al., “Probabilistic design verification”, *Proc. International Conference on Computer-Aided Design*, 1991, pp. 468-471.

[20] P. Jain and G. Gopalakrishnan, “Efficient symbolic simulation-based verification using the parametric form of boolean expressions”, *IEEE Transactions on CAD*, Vol. 13, pp. 1005-1015, August 1994.

[21] S. Kang and S. A. Szygenda, “The simulation automation system (SAS); concepts, implementation, and results”, *IEEE Transactions on VLSI Systems*, Vol. 2, pp. 89-99, March 1994.

[22] H. Kim and J. P. Hayes, “High-coverage ATPG for datapath circuits with unimplemented blocks”, *Proc. International Test Conference*, 1998, pp. 577–586.

[23] K. N. King and A. Jefferson Offutt, “A Fortran language system for mutation-based software testing”, *Software Practice and Experience*, Vol. 21 (7), pp. 685-718, 1991.

[24] W. Kunz, D. K. Pradhan, and S. M. Reddy, “A novel framework for logic verification in a synthesis environment”, *IEEE Transactions on CAD*, Vol. 15, pp. 20-32, January 1996.

[25] W. Kunz, D. Stoffel, and P. R. Menon, “Logic optimization and equivalence checking by implication analysis”, *IEEE Transactions on CAD*, Vol. 16, pp. 266-281, March 1997.

[26] S. Kuo, “Locating logic design errors via test generation and don't care propagation”, *Proc. European Design Automation Conference*, 1992, pp. 466-471.

[27] H. K. Lee and D. S. Ha, “On the generation of test patterns for combinational circuits”, Dept. of Elec. Eng., Virginia Tech., Rep. 12-93, 1993.

[28] G. Odawara et al., “A logic verifier based on boolean comparison”, *Proc. Design Automation Conference*, 1986, pp. 208-214.

[29] C. Pixley et al., “Commercial design verification: Methodology and tools”, *Proc. International Test Conference*, 1996, pp. 839-848.

[30] I. Pomeranz and S. M. Reddy, “On error correction in macro-based circuits”, *Proc. International Conference on Computer-Aided Design*, 1994, pp. 568-675.

[31] B. Rohfleisch, B. Wurth, and K. Antreich, “Logic clause analysis for delay optimization”, *Proc. Design Automation Conference*, 1995, pp. 668-672.

[32] D. Siewiorek and R. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, Burlington, Mass., 1992.

[33] Texas Instruments, *The TTL Logic Data Book*, Dallas, 1988.

[34] R. Wei and A. Sangiovanni-Vincentelli, “PROTEUS: A logic verification system for combinational circuits”, *Proc. International Test Conference*, 1986, pp. 350-359.

[35] M. Yoeli (ed.), *Formal Verification of Hardware Design*, IEEE Computer Society Press, Los Alamitos, Calif., 1990.

Hussain Al-Asaad is an assistant professor in the department of Electrical and Computer Engineering at the University of California, Davis. His research interests include design verification, testing, and fault tolerant computing. Al-Asaad received his B.E. (with distinction) in Computer and Communications Engineering from the American University of Beirut, Lebanon, his M.S. in Computer Engineering from Northeastern University, Boston, and his Ph.D. in Computer Science and Engineering from the University of Michigan, Ann Arbor. He is a member of IEEE, ACM, and Sigma Xi.

John P. Hayes is Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, where he teaches and conducts research in the areas of computer-aided design, verification and testing; VLSI circuits; and embedded system architectures. He received the B.E. degree from the National University of Ireland, Dublin, and the M.S. and Ph.D. degrees from the University of Illinois, Urbana-Champaign, all in electrical engineering. At the University of Illinois he participated in the design of the ILLIAC III computer. In 1970 he joined the Shell Benelux Computing Center in The Hague, where he worked on mathematical programming. From 1972 to 1982 Dr. Hayes was a faculty member of the Departments of Electrical Engineering–Systems and Computer Science of the University of Southern California, Los Angeles. He joined the University of Michigan in 1982. He was the founding director of the University of Michigan's Advanced Computer Architecture Laboratory. He served as Technical Program Chairman of the 1977 International Conference on Fault-Tolerant Computing, and the 1991 International Computer Architecture Symposium. Dr. Hayes is the author of numerous technical papers and five books, including *Layout Minimization of CMOS Cells*, (Kluwer, 1992; coauthored with R. L. Maziasz), *Introduction to Digital Logic Design*, (Addison-Wesley, 1993), and *Computer Architecture and Organization*, (3rd ed., McGraw-Hill, 1998). He has served as editor of various technical journals, including the *IEEE Transactions on Parallel and Distributed Systems* and the *Journal of Electronic Testing*. Dr. Hayes is a Fellow of IEEE and a member of ACM and Sigma Xi.