Universal Fault Simulation using Fault Tuples

Kumar N. Dwarakanath and R. D. (Shawn) Blanton
Center for Electronic Design Automation
Carnegie Mellon University, Pittsburgh PA 15213
{nari, blanton}@ece.cmu.edu

ABSTRACT

We introduce a new fault representation mechanism for digital circuits based on fault tuples. A fault tuple is a simple 3-element condition for a signal line, its value, and clock cycle constrain t. AND-OR expressions of fault tuples are used to represent arbitrary misbehaviors. A fault simulator based on fault tuples was used to conduct experiments on benc hmark circuits. Simulation results show that a 17% reduction of average CPU time is achiev ed when performing sim ulation on all fault types simultaneously, as opposed to individually. We expect further improvements in speedup when the shared characteristics of the various fault types are better exploited.

1. INTRODUCTION

The classical single stuck-line (SSL) fault model is the most commonly used fault model in digital systems testing. The SSL fault model assumes that any single circuit line is susceptible to a permanent stuck-at logic v alue of 0 or 1. Among the many reasons for its contin ued appeal are: (a) the n umber of SSL faults is linearly related to the number of lines in the circuit, (b) the model maps well to the gate level, (c) the SSL tests have traditionally done a good job of detecting non-SSL misbehavior, and (d) there exist many commercial test generation and fault sim ulationtools based on single stuck-line misbehavior.

Ho w ever, previous w rk has shown that real defects do not beha ve like SSL faults, thus making tasks like diagnosis difficult. Moreover, the SEMATECH experiment [6] and other similar experiments [11] ha veshown that SSL tests alone are not sufficient for obtaining high defect coverage. As a result, other fault models that accurately reflect realistic circuit failures ha ve been explored [2] [13] [14] [15] [3]. In addition, approaches lik e inductive fault analysis [9] [7], specify a realistic fault set by investigating the physical defects that lead to a failure. Stuc k-at tests are therefore, typically augmented b y tests that target other fault t ypes (bridging, delay, *etc.*) in order to improve defect co verage. It is likely that this trend of augmenting stuc k-at tests with other t ypes of tests will continue due to the on-going changes in tec hnology.

In an ticipationof the abo ve situation,w e areproposing a fault modeling mechanism that allows many arbitrary misbeha viors to be represented using expressions of primitives w e call*fault tuples*. A fault tuple is a 3-element subfault providing conditions for a signal line, its value, and clock cycle constrain t. *AND-OR* expressions of fault tuples, that we call *macrofaults*, can be used to represent various misbehaviors. The use of fault tuples allow both existing and emerging fault models to be represen tedusing one common mechanism. The main advan tages of this common representation are (a) simultaneous analysis of different misbehaviors, (b) exploitation of common information among various misbeha viors, and (c) a single method for analyzing misbehaviors.

In another approach [12], the authors present a library-based modeling mechanism used for the simulation of non-classical faults at the gate level. F or a giv en cell library , the effects of realistic defects are investigated. These effects are mapped to gate-lev el faults and stored in a fault library F or a giv en circuit under test, a realistic target fault set is obtained from the library and fault simulation is performed on this fault set. The main drawback of this approach is that the misbeha viors considered are limited by the fault effects stored in the fault library.

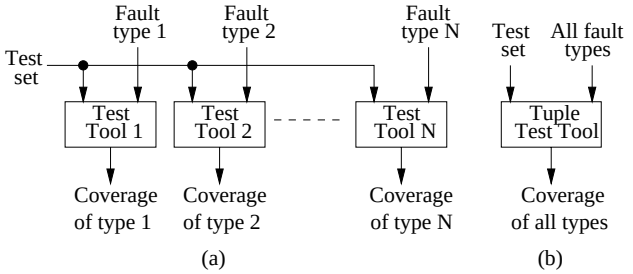

Figure 1: (a) Current point tools used for test analysis (*i.e.* fault simulation, A TPG, *etc.*) for various fault types versus (b) our proposed method for primitive fault tuples.

Due to the different misbehaviors exhibited by various fault types, a separate test tool is typically constructed for each fault model. By using the fault tuple mechanism for representing fault t ypes, one comprehensiv tool can be used to

DAC 2000, Los Angeles, California

simultaneously perform fault simulation and test generation for many types of faults as illustrated in Figure 1.

The rest of the paper is organized as follows. In Section 2, fault tuples are defined and notation is introduced. Section 2 also illustrates how fault tuples can be used to represent various misbehaviors. In Section 3, the fault simulation methodology is presented and in Section 4, we present our experiment methodology and fault simulation results using our tuple-based fault simulator. Finally, we draw conclusions and describe future work in Section 5.

2. FAULT TUPLES

A *fault tuple*[1] f is defined as a 3-tuple represented as $<l, v, t>$, where l is a signal line, v is a value, and t is a clock cycle constraint. The value set for each of the elements is given as follows:

$$\begin{aligned} l &\in \{lines\} \\ v &\in \{0, 1, D, \overline{D}\} \\ t &\in \{i, i+N, i>, i\geq, ...\} \end{aligned}$$

A tuple $f = <l, v, t>$ is *satisfied* if and only if the signal line l is controlled to the value v in a clock cycle described by t and the corresponding error (if any) is propagated to an observable point. The tuple elements l and v and their corresponding value sets are well-known, however the element t needs elaboration. If $v \in \{0, 1\}$, then the signal l must be controlled to v within the clock cycle range described by t. For example, $t = i$ means that the tuple is satisfied if l is controlled to v in any clock cycle i. If $v \in \{D, \overline{D}\}$, then t describes the range of clock cycles where a signal line must be activated and the resulting error discrepancy (D or $\overline{D}$) manifests. Satisfaction of a tuple in this case also requires that the error be propagated to an observable point anytime in the future. The value $t = i + N$, where N is some integer, means the tuple must be satisfied in the Nth clock cycle after some reference clock cycle i. Satisfaction must occur in a clock cycle j before a reference clock cycle i if $t = i<$. Other t values such as $\{>, \geq, \leq, \neq\}$ are similarly explained. The values assignable to the tuple element t_j for a particular tuple f_j are variables themselves that can be instantiated either by another tuple or set to a particular value N by the user.

Combinations of tuples allow the representation of arbitrary misbehaviors. A *product* P is defined as an *AND* expression of tuples. For example, product $P_1 = <f_1><f_2>$ is an *AND* expression of the tuples $<f_1>$ and $<f_2>$. A product is satisfied if and only if all the tuples within the product are satisfied[2]. Thus, $P_1 = <f_1><f_2>$ is satisfied only if both tuples $<f_1>$ and $<f_2>$ are satisfied. A *macrofault* M is an *OR* expression of products, and therefore is an *AND-OR* expression of tuples. The macrofault $M_1 = \{<f_1><f_2> \mid <f_3><f_4>\}$ is a disjunctive expression of the products $P_1 = <f_1><f_2>$ and $P_2 = <f_3><f_4>$, where the symbol $\mid$ is used to denote the *OR* relation between products. A macrofault M is detected if and only if one or more of its products are satisfied. Thus, M_1 is detected if and only if the product $P_1 = <f_1><f_2>$ is satisfied or product $P_2 = <f_3><f_4>$ is satisfied.

[1] A fault tuple will simply be referred to as tuple in the rest of the paper.

[2] A product that contains more than one tuple with $v \in \{D, \overline{D}\}$ is satisfied if one or more of the discrepancies are made observable.

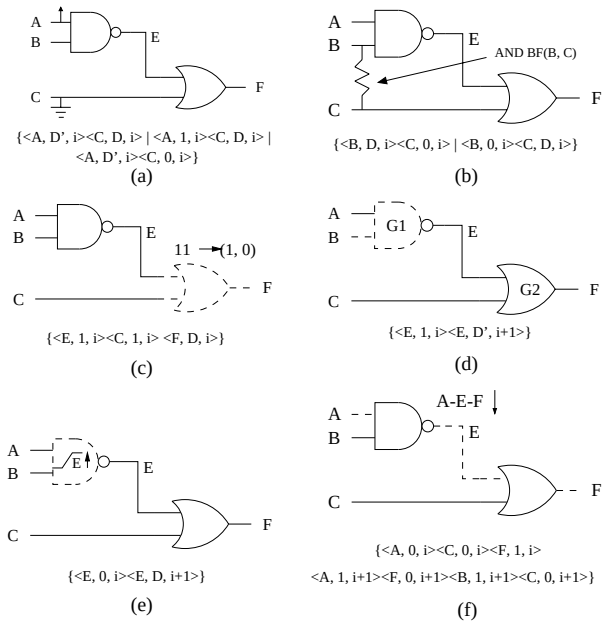

Figure 2: Various fault types described using fault tuples: (a) MSL fault (A/1, C/0); (b) AND-bridging fault between lines B and C; (c) OR-gate pattern fault 11→(1,0); (d) N-transistor of gate G1 connected to line B is stuck-open; (e) Slow-to-rise NAND gate transition fault E↑; (f) Slow-to-fall robust path delay fault AEF↓.

The fault tuple is fault modeling mechanism that allows for the representation of a significant number of fault types. Consider an SSL fault $f = a/1$ (*i.e.* signal line a is permanently stuck-at 1). This can be represented using the tuple format as: $\{<a, \overline{D}, i>\}$, which indicates that fault f is detected if in any clock cycle i, a value 0 is applied to line a and the error discrepancy $\overline{D}$ due to f is propagated to an observable point. These are exactly the conditions for detecting the SSL fault f. Other fault types can also be easily represented. Figures 2a, 2b and 2c illustrate how the tuple mechanism can be used to represent a multiple stuck-line (MSL) fault, an AND-bridging fault and a pattern fault [8], respectively. The MSL and AND-bridging faults can be detected in multiple ways as indicated by the disjunction of the tuple product expressions. For example, the MSL fault $M = \{A/1, C/0\}$ is detected if and only if the faulty circuits with both SSL faults $A/1$ and $C/0$ is detected, or $A = 1$ and $C/0$ is detected, or $A/1$ is detected and $C = 0$. The use of the same clock cycle variable $t = i$ for the MSL and bridging faults indicates that the tuples are not independent, that is, all the tuples of a product have to be satisfied in the same clock cycle i. Transistor stuck-open (TSO) faults [15] can also be represented using the tuple representation mechanism (Figure 2d).

Timing or dynamic faults can also be easily captured by the tuple mechanism. Figures 2e and 2f illustrate transition and

robust path delay faults, respectively. These macrofaults use relational values, like the TSO fault of Figure 2d, for the tuple element t. The slow-to-rise NAND gate transition fault (Figure 2e) is detected if the transition is intialized ($E = 0$) by first test vector and the slow transition represented by the error discrepancy D appearing on line E is propagated to an observable point by the second test vector. This sequence of test vectors is captured by the time values i and $i + 1$.

From the expressions for the macrofaults, it can be observed that tuple sharing exists among various fault types. For example the tuple $<C, D, i>$ is present in the expression for the MSL and the AND-BF macrofaults of Figure 2. Tuple sharing is exploited by tracking the corresponding macrofaults when tuples are analyzed (simulated).

3. FAULT TUPLE SIMULATION

A fault simulator based on the fault tuple modeling mechanism has been developed. The core of the fault simulator is based on the concurrent event-driven fault simulation method [1]. Figure 3 shows the pseudocode for the complete fault simulation algorithm. A fault simulation begins by identifying an initial set of tuples from an input list of macrofaults. After simulation of each test vector, the macrofault list is updated based on the set of tuples satisfied (Figure 4). Specifically, macrofault update determines if any

```
fault_tuple_simulation(C, V, M);
/* C = Circuit under test */
/* V = Test vectors for simulation */
/* M = Macrofaults to be simulated */
begin
    curr_sim_id = 1
    /* Form the initial list of tuples to be simulated */
    /* Macrofaults added for update */
    initial_list_tuples(M, curr_sim_id)
    for v: all test vectors of V
        /* Perform concurrent fault simulation of tuples */
        concurrent_fault_sim(v)
        /* Update macrofaults using tuple satisfaction */
        for m: all macrofaults to be updated
            /* For the current clock cycle */
            update_macrofault(m, curr_sim_id)
        end for on m
    end for on v
    report macrofault coverage
end
```

Figure 3: Pseudocode for performing fault simulation using the fault tuple representation mechanism.

macrofault has been detected or if additional tuples have to be simulated for macrofault detection. In order to facilitate macrofault update, a product P_j is further broken into states, where the number of states for P_j is equal to the number of different time element values found in P_j. For example, $P_j = <a, 0, i><b, 0, i><a, D, i + 1>$ contains two states (i and $i + 1$).

4. FAULT SIMULATION RESULTS

A fault simulator based on fault tuples, called **FaTSim**, has been implemented with approximately 5000 lines of C code. FaTSim currently performs fault simulation of combinational circuits only. It was used to perform experiments on the ISCAS85 benchmark circuits [4], a 64-bit ALU, and part of the CMU56K DSP ITC'99 benchmark circuit [5](*i.e.* the

```
update_macrofault(M, PID, SID);
/* M = Macrofault to be updated */
/* PID = Identification no. (ID) for product P ∈ M */
/* SID = ID for state S ∈ P */
begin
    prod = product of M with product PID
    s = tuples of prod with SID
    detected = TRUE /* Initialize flag */
    while (detected == TRUE)
        for f: all fault tuples of s
            if (f is a tuple with v ∈ {D, D̄})
                /* Check if tuple's faulty machine is satisfied */
                if (f is not satisfied)
                    detected = FALSE
                    break
            else /* A tuple with v ∈ {0, 1} */
                /* Check the value of the line given in f */
                if (f is not satisfied)
                    detected = FALSE
                    break}
                end if
            end if
        end for on f
    end while
    if (detected is TRUE)
        s = tuples of prod with ID=SID+1
        if (s is NULL)
            /* No more states left so, macrofault detected */
            mark macrofault detected
            return
        else {/* There are more states */
            /* Add tuples from state for fault simulation */
            add tuples with ID=SID+1 for simulation
            schedule_macrofault_for_update(M, PID, SID+1)
        end if
    else /* Some tuples were not satisfied */
        add tuples with SID=0 for simulation
        schedule_macrofault_for_update(M, PID, 0)
    end if
end
```

Figure 4: Pseudocode for updating a macrofault.

largest combinational portion of the data-path). To demonstrate efficiency of performing fault simulation using the fault tuple mechanism, the following fault types were analyzed: SSL, AND-NFBF (non-feedback bridging), TSO and MSL. More specifically, we used a collapsed set of SSL faults, TSO faults generated by the test generator SOPRANO [10], and a randomly-chosen set of MSL and AND-NFBF faults. Test sets generated by SOPRANO were used for fault simulation, and all experiments were performed on a 300MHz Ultra SPARC-II SUN workstation with 1.0 GB of memory.

Table 1 shows the CPU execution times for performing fault simulation using FaTSim. First, each fault type was fault simulated individually using test sets of the size listed in column 2. The sum of the individual execution times is shown under the column "Total". All the fault types were then fault simulated together using the same test set. The CPU execution time for this one comprehensive fault simulation is shown in the "All" column. Note, the fault coverage of the individual simulations match that of the comprehensive simulation.

As indicated in Table 1, the fault simulation time for all the faults types together shows a 10-37% (an average of 17% for the benchmarks considered) reduction in CPU time com-

Benchmark name	No. of test vectors	% common tuples[4]	SSL CPU time (sec.)	AND-NFBF CPU time (sec.)	TSO CPU time (sec.)	MSL CPU time (sec.)	Total (sec.)	All (sec.)	% CPU time reduction[3]
c432	145	26.22	0.36	0.26	0.55	0.13	1.30	0.81	38.0
c499	205	15.93	0.75	0.33	0.91	0.21	2.20	1.97	10.5
c880	189	27.57	1.25	0.72	1.77	0.47	4.21	3.13	25.7
c1355	345	32.57	2.71	3.05	3.73	2.10	11.59	9.58	17.3
c1908	393	28.28	4.10	5.08	7.43	33.97	50.58	39.90	21.1
c2670	397	30.83	5.11	3.83	9.35	2.65	20.94	17.38	17.0
c3540	685	32.51	18.63	12.13	49.50	18.07	98.33	85.23	13.3
c5315	608	33.18	15.92	12.01	37.90	12.42	78.25	70.28	10.2
c6288	252	27.10	19.51	7.78	26.43	13.06	67.14	58.53	12.8
c7552	795	30.84	30.45	23.15	64.15	188.35	306.10	266.41	13.0
alu64	516	30.01	28.33	19.20	51.17	94.71	193.41	167.11	13.6
CMU56K	810	31.98	60.13	50.42	99.90	53.83	266.26	234.9	11.8

Table 1: CPU execution times (in seconds) for fault simulation of various fault types for benchmark circuits.

pared to the sum of the individual times[3]. We believe this is due to the fact that there exists common tuple information (column 3)[4] among the various fault types.

5. CONCLUSIONS

Obtaining an accurate defect coverage typically means that SSL test patterns have to be augmented with tests that are aimed at other fault types. In this work, instead of proposing another fault model, we have developed a new fault modeling mechanism that allows arbitrary misbehaviors to be represented as AND-OR expression of simple faults, we call fault tuples.

Using the fault tuple mechanism, we demonstrated fault simulation of various fault types within one comprehensive tool. Exploitation of common information among various fault types resulted in a 17% average reduction of CPU run times. We expect a more significant speedup when: (1) macrofaults are represented using minimal information and, (2) collapsing is performed across macrofaults of various types.

Future work will focus on developing theory for fault collapsing across macrofaults of various types, and adding capability for handling sequential circuits. In addition, we will also explore ways of using the fault tuple mechanism for high-level testing.

6. REFERENCES

[1] M. Abromovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, Piscataway, NJ, 1990.

[2] P. Banerjee and J. A. Abraham. Characterization and testing of physical failures in MOS logic circuits. *IEEE Design & Test of Computers*, Vol. 1, No. 3, pp. 76–86, Aug. 1984.

[3] R. D. Blanton and J. P. Hayes. Properties of the input pattern fault model. In *Proc. of 1997 International Conference on Computer Design*, pp. 372–80, Oct. 1997.

[4] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN. In *Proc. 1985 International Symposium on Circuits and Systems*, pp. 695–8, June 1985.

[5] S. Davidson. Panel 6: ITC'99 benchmark circuits-preliminary results. In *Proc. of 1999 International Test Conference*, pp. 1125, Sept. 1999.

[6] P. Nigh et al. An experimental study comparing the relative effectiveness of functional, scan, I_{ddq} and delay fault testing. In *Proc. of 1997 VLSI Test Symposium*, pp. 459–64, April 1997.

[7] F. J. Furguson and J. P. Shen. Extraction and simulation of realistic CMOS faults using inductive fault analysis. In *Proc. of 1988 International Test Conference*, pp. 475–84, Sept. 1988.

[8] IBM Inc. *TestBench Library Data Reference*, Nov. 1996.

[9] J. Khare and W. Maly. *From Contamination to Defects, Faults and Yield Loss: Simulation and Applications*. Kluwer, Norwell, MA, 1996.

[10] H. K. Lee and D. S. Ha. Soprano: An efficient automatic test pattern generator for stuck-open faults in CMOS combinational circuits. In *Proc. of 27th ACM/IEEE Design Automation Conference*, pp. 660–6, 1990.

[11] S. C. Ma, P. Franco, and E. J. McCluskey. An experimental chip to evaluate test techniques experiment results. In *Proc. of 1995 International Test Conference*, pp. 663–72, Oct. 1995.

[12] U. Mahlstedt and J. Alt. Simulation of non-classical faults on the gate level: The fault simulator COMSIM. In *Proc. of 1993 International Test Conference*, pp. 883–92, Oct. 1993.

[13] K. C. Y. Mei. Bridging and stuck-at faults. *IEEE Transactions of Computers*, Vol. 23, No. 7, pp. 720–7, July 1974.

[14] G. L. Smith. A model for delay faults based upon paths. In *Proc. of 1985 International Test Conference*, pp. 342–9, Nov. 1985.

[15] R. L. Wadsack. Fault modeling and logic simulation of CMOS and MOS integrated circuits. *Bell System Technical Journal*, Vol. 57, No. 5, pp. 1449–74, May-June 1978.

[3] Initialization times (parsing, data structure creation, *etc.*) are not included in the simulation times.

[4] The percentage of common tuples is calculated by dividing the number of shared tuples by the total number of tuples.