A Hybrid Approach to Equivalent Fault Identification for Verification Environment Qualification

Chia-Cheng Wu
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Tung-Yuan Lee
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Yung-An Lai
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Hsin-Pei Wang
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

De-Xuan Ji
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Yan-Ping Chang
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Teng-Chia Wang
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Chin-Heng Liu
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Chun-Yao Wang
National Tsing Hua University,
Hsincu, Taiwan, R.O.C.

Yung-Chih Chen
Yuan Ze University,
Chungli, Taiwan, R.O.C.

ABSTRACT

Fault-based verification technique is a method to qualify a verification environment. The better verification environment can detect output differences between the fault-free and fault-injected circuits with a higher probability. Since different injected faults could cause the same output response under all stimuli, which are called equivalent faults, maximally identifying these equivalent faults can improve the efficiency of verification environment qualification without sacrificing its quality. The 2016 CAD Contest at ICCAD posed the problem of identifying equivalent faults in the circuits. This paper presents our work in the Contest with some improvements.

KEYWORDS

Verification environment qualification; fault injection; equivalent fault identification; mandatory assignment (MA)

ACM Reference Format:
Chia-Cheng Wu, Tung-Yuan Lee, Yung-An Lai, Hsin-Pei Wang, De-Xuan Ji, Yan-Ping Chang, Teng-Chia Wang, Chin-Heng Liu, Chun-Yao Wang, and Yung-Chih Chen. 2018. A Hybrid Approach to Equivalent Fault Identification for Verification Environment Qualification . In *Proceedings of Great Lakes Symposium on VLSI 2018 (GLSVLSI'18).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3194554.3194635

1 INTRODUCTION

Functional verification is a task to confirm the consistency between the implementation and specification. When designs are getting more complex, the process of functional verification takes more time and effort. Simulation-based verification [15] has been a common practice in verification community. However, due to the fact that exhaustive simulation is infeasible for larger designs, *coverage metrics* have been proposed to measure the quality of verification and thus reduced the simulation cost.

Structural coverage metrics [8][13], also known as *code coverage metrics*, are such techniques. However, most structural coverage metrics only focus on activation of the design content from the stimuli, but do not consider the abilities of stimuli to propagate the error effects to observation points or Primary Outputs (POs). Therefore, the higher coverages in these metrics do not always imply the better quality of the verification environments. Thus, *Fault-based verification* technique [7][1][4][2], which is an effective approach for evaluating the quality of the verification environment, has been proposed afterward.

By injecting artificial faults into the original implementation, designers can examine whether the verification environment differentiates the faulty and fault-free designs. If all the injected faults can be detected, it indicates that the verification environment is robust and effective to reveal errors; otherwise, the verification environment contains some weakness points for improvement. Thus, the fault-based verification methodology *does* deal with the error effect propagation issue that the structural coverage metrics do not.

Mutation analysis is a fault-based verification technique originating from the software engineering [17]. Based on two Hypothesis: the *Competent Programmer Hypothesis* and *Coupling Effect Hypothesis*, mutation analysis only targets at a subset of all potential faults and assumes that these faults are sufficient to represent all faults. Competent Programmer Hypothesis states that programmers develop their programs very close to the correct version. Coupling Effect Hypothesis assumes that complex faults can be coupled by simple faults such that a test set detecting all the simple faults in a program will detect a very high percentage of the complex faults [6].

A commercial EDA tool, Certitude^{TM} [12][19], implements the mutation analysis approach, where RT-level designs are usually used for testbench qualification. Recently, *ISO 26262* [16], which is standard for defining functional safety of electrical and/or electronic systems of automobiles, has been deployed. In this standard, artificial faults are injected into gate-level designs [10][9][11][16]

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 106-2221-E-007-111-MY3, MOST 106-2221-E-155-056, MOST 103-2221-E-007-125-MY3, and MEDIATEK Research Center Doctoral Talent Fellowship

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GLSVLSI'18, May 23–25, 2018, Chicago, IL, USA
© 2018 Association for Computing Machinery.
ACM ISBN ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194635

to see whether the verification environments can detect these faults or not.

Since the injected faults at different locations could cause the same output response under all stimuli, which are called *equivalent faults*, only one of these equivalent faults need to be verified within the verification environment. Thus, maximally identifying these equivalent faults reduces the number of faults to be injected, and improves the efficiency of a verification environment qualification, i.e., the number of verification patterns is reduced without sacrificing its quality. The 2016 CAD Contest at ICCAD [18] posed such problem of identifying equivalent faults, i.e., the faults having the same fault effect, in the circuits. Its formulation is as follows: Given a set of fault models and the original netlist of design consisting of two-input gates, to identify the equivalent faults maximally in the netlist for elevating the verification efficiency under a single fault injection mechanism.

In this paper, we present a hybrid approach to identify the equivalent faults efficiently. The approach consists of four steps, and they are fault collapsing, Mandatory Assignment (MA) calculation, structural matching, and redundant fault identification. To complete the description of the work, we will introduce all the steps in the paper. However, the MA calculation and structural matching steps, which are the main contributions of this work, will be emphasized. Furthermore, from the experimental perspective, these two steps also identify many equivalent faults within a few seconds.

2 PRELIMINARIES

2.1 Fault Model

11 types of fault models in the Contest are divided into three classes.

2.1.1 Stuck-at fault. The faulty wire is set to 0 or 1, and denoted as SA0 and SA1, respectively.

2.1.2 Negated fault. The faulty wire is inverted and denoted as NEG.

2.1.3 Gate-replacement fault. This fault changes the original driver gate type of a faulty wire. The faults in this class are denoted as RDOB_G, where G = { AND, NAND, OR, NOR, XOR, XNOR, NOT, BUFF}.

2.2 Equivalent Fault

With the fault models, a faulty circuit is derived by injecting a single fault into the original circuit. If two faulty circuits have the same output response under all input stimuli, these faults are equivalent faults.

3 EQUIVALENT FAULT IDENTIFICATION

This section presents the proposed algorithm for equivalent fault identification. The algorithm consists of four steps and they are fault collapsing, MA calculation, structural matching, and redundant fault identification.

3.1 Fault Collapsing

The fault collapsing step is only applied to the stuck-at fault model within a fanout-free region. This is because the fault-effect of a single fault in the fanout-free region remains singularly. The idea of fault collapsing is that when the fault-effect of two distinct stuck-at faults are the same, they are equivalent faults. For example, in Fig. 1, the fault-effect at a is 1/0 (fault-free value/faulty value), and b has to be assigned 1 to propagate the fault-effect 1/0 to c. For the output c SA0 fault, the fault-effect at c is also 1/0, and b is also 1. Thus, a SA0 fault and c SA0 fault have the same fault-effect and they are equivalent faults.

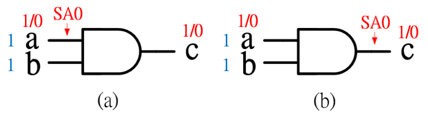

Figure 1: Equivalent faults for an AND gate.

3.2 MA Calculation

The MA is the unique value assignment to a wire necessary for a test to exist. The *logic implication* is a process of computing MAs for a test. The MAs for a test on a wire w can be computed by setting the fault-activating value or setting the noncontrolling values on the side inputs of w's propagating path. Then, these assignments can be propagated forward or backward to obtain more MAs. Recursive learning [14], can be used to perform logic implications more completely. If the MAs of one fault are inconsistent, the fault is untestable.

To detect the considered fault models, we have to derive their test patterns. Different faults usually have different sets of test patterns. For two distinct but equivalent faults, however, their test pattern sets are identical. To express the test pattern set for a fault, we can explicitly enumerate every test pattern. However, choosing this way might be time-consuming. Thus, to express the test pattern set of a fault compactly, we can use the idea of MA. This is because MAs are the unique value assignments to wires necessary for a test pattern to exist. That is, MAs are the common value assignments to wires among all the test patterns of the fault. Unfortunately, computing all the MAs for detecting a fault, which is equivalent to deriving all the test patterns of a fault, is an NP-hard problem [3]. Thus, we propose another method to determine the equivalent faults based on a partial set of MAs as well as other information, which will be explained in detail in the following paragraphs.

In the process of MA calculation, sometimes we cannot determine a unique value assignment for a fanin node from a known MA at its output. Since these values are not unique for each fanin node, they are not MAs. We name these fanin pairs *active pairs*. For another example, in Fig. 2(a), to activate the fault-effect of $Fault_{g,RDOB\_NAND}$, the fanin nodes (h, i) = {(1, 1), (0, 1), (1, 0), (0, 0)}. This is because these value assignments can differentiate the NAND and AND from their output values. However, to propagate the fault-effect, we have to set (h, i) = (1, 1). As a result, only the active pair (h, i) = (1, 1) is kept, and h = 1, i = 1 are MAs. Furthermore b = 0 is an MA, and (j, k) = {(0, 1), (1, 0), (1, 1)} are active pairs by backward logic implications from h and i, respectively.

To determine the equivalent faults, it is not necessary to derive the test patterns for each fault. Here we propose the idea of *fault-effect influential region*. When injecting a fault into a circuit, certain subcircuit is influenced by the fault. We can construct the fault-effect influential region for representing a fault. This region is constructed in forward and backward directions. In the forward direction, it is from the fault site to the fanout node of the fault-propagating path or POs. In the backward direction, it is from the fault site to the active pairs, the wire with MAs, or PIs. To determine if two faults are equivalent or not, we can construct the fault-effect influential region for each fault first. If the boundaries of these two regions are identical, and the values on the boundaries are the same, these two faults are equivalent faults. This is because the circuit structures surrounding the fault-effect influential regions are exactly identical. Thus, the fault-effects propagation from these regions to the POs are the same. Theorem 1 is used for supporting this equivalent fault identification.

Theorem 1 : Given two faults, if the boundaries of their fault-effect

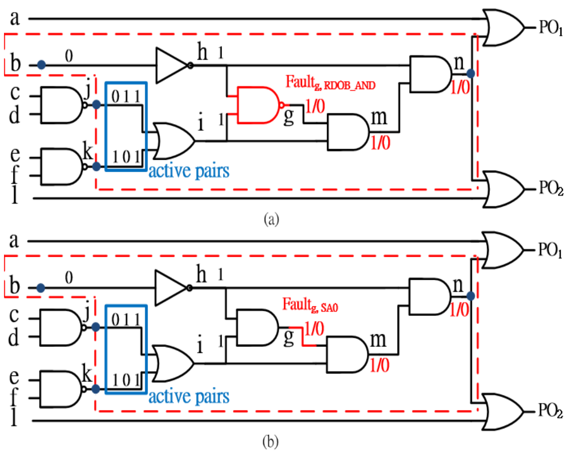

Figure 2: An example of calculating MAs and active pairs of two distinct faults injected into the same circuit. (a) An RDOB_NAND fault occurs at g. (b) A SA0 fault occurs at g.

influential regions are identical, and the values on the boundaries are the same, these two faults are equivalent.

For example, in Fig. 2, consider injecting two distinct faults at g $Fault_{g,RDOB\_NAND}$ and $Fault_{g,SA0}$ in Fig. 2(a) and Fig. 2(b). The dashed regions are their fault-effect influential regions. Since the boundaries of the regions are identical and the values on the boundaries are the same, these two faults are equivalent faults.

3.3 Structural Matching

Since computing all the MAs for a fault is computationally intensive, the MAs derived in the last step might be incomplete for cost and performance tradeoff. Thus, not every equivalent fault pair can be identified. However, we can use structural properties to identify additional equivalent faults. Since XOR and XNOR gates do not have MAs on the side inputs, this step is especially useful for them within fanout-free structures.

Next, we propose some conditions that assert the equivalence of two distinct faults.

Condition 1 : If the structures of two faulty circuits are identical, these two faults are equivalent faults.

Condition 2 : For an XOR or XNOR gate within a fanout-free structure, its input NEG fault and the output NEG fault are equivalent faults.

Definition 1 : A *fanout-free chain* is composed by two gates with a Common Side Input (CSI), and the other two inputs, a, b, are fanout-free as shown in Fig. 3. If only b is fanout-free, the chain is called a *loosened fanout-free chain.*

According to the locations of equivalent faults in a fanout-free chain, we further classify the equivalent faults into two classes and they are summarized in TABLE 1. Here we only discuss one case for class 1 and 2 in Condition 3-1 and Condition 3-2, respectively.

Condition 3-1 : For gate B = XOR and gate C = XOR in the fanout-free chain, a SA1 (SA0) fault at a and a SA1 (SA0) fault at c are equivalent.

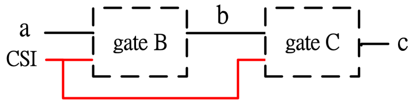

Figure 3: Fanout-free chain.

When an input of an XOR gate is determined, the XOR gate becomes a BUFF or NOT. Hence, we discuss this with a fixed CSI value. In Fig. 3, since a, b are fanout-free, if the CSI is 1, the circuit becomes two connected NOT gates, and the fault-effect at a can be only propagated to c. Similarly, if CSI is 0, the circuit becomes two connected BUFF gates, and the fault-effect at a can be only propagated to c, too. Thus, the fault-effect at c is the same as that at a. That is, $Fault_{a,SA0}$ = $Fault_{c,SA0}$ and $Fault_{a,SA1}$ = $Fault_{c,SA1}$. The other three cases in class 1 can be explained in a similar way.

Condition 3-2 : For gate B = XOR and gate C = XNOR in the loosened fanout-free chain, an RDOB_AND at b and an RDOB_NAND at c are equivalent.

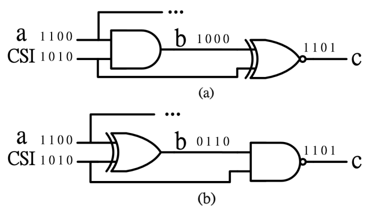

Figure 4: An example for Condition 3-2. (a) RDOB_AND at b. (b) RDOB_NAND at c.

Table 1: Equivalent faults by structural matching.

class	gate B	gate C	equivalent faults
1	**XOR**	**XOR**	**a SA0 = c SA0, a SA1 = c SA1**
	XNOR	XNOR	a SA0 = c SA0, a SA1 = c SA1
	XOR	XNOR	a SA0 = c SA1, a SA1 = c SA0
	XNOR	XOR	a SA0 = c SA1, a SA1 = c SA0
2	**XOR**	**XNOR**	**b RDOB_AND = c RDOB_NAND**
	XNOR	XOR	b RDOB_OR = c RDOB_NOR
	OR	XNOR	b SA0 = c RDOB_NAND
	OR	XOR	b SA0 = c RDOB_AND
	AND	XNOR	b SA1 = c RDOB_OR
	AND	XOR	b SA1 = c RDOB_NOR

3.4 Redundant Fault Identification

After running the first three steps, we group equivalent faults into a fault group. However, there might be groups of faults that are all redundant faults, and should be merged into one group. Next, in this step, we use SAT solvers to merge groups of redundant faults together.

Instead of comparing each fault group with the fault-free circuit exhaustively, we propose a heuristic to elevate the efficiency. We observed that the local functional difference between certain faulty circuit and the fault-free one is very small. This value assignment could be *Satisfiability Don't Cares* (SDCs) of faulty circuit. If this happens, the fault is a redundant fault. For example, when an RDOB_XOR fault replaces an OR gate, the local functional difference between them is only input pair (1, 1). We call this fault a *redundant fault candidate*. Then, we construct a Miter [5] from this faulty circuit and the fault-free circuit and use SAT solvers to verify if this redundant fault candidate is redundant or not. Using this way, only the fault groups containing a redundant fault candidate are considered for checking by SAT solvers, and the efficiency of this step is elevated.

4 EXPERIMENTAL RESULTS

The proposed algorithm was implemented in C++ and the experiments were conducted on the official platform of the Contest. In the experiment, we used the benchmarks which came from the 2016 CAD Contest [18] and compared the results with top three teams in the Contest. The Contest used 4 hidden benchmarks, Case05 to

Table 2: The experimental results on 2016 CAD Contest benchmarks.

Bench.	|PI|	|PO|	|gate|	Init.	Gold.	Team A			Team B			Team C			Ours			100%
						|gp|	CPU	Cov.	|gp|	CPU	Cov.	|gp|	CPU	Cov.	|gp|	CPU	Cov.	CPU
Case05	46	43	401	3098	2029	2044	0.4	98.5	2029	0.6	100	2029	1.0	100	2029	9.2	100	1221
Case06	13	14	648	5008	1020	1620	28.6	84.9	1020	41.6	100	1132	13.1	97.1	1132	4.4	97.1	3808
Case07	17	18	567	4352	1238	1870	14.9	79.7	1770	66.0	82.9	1492	41.2	91.8	1480	15.8	92.3	4011
Avg.	-	-	-	-	-	-	-	87.7	-	-	94.3	-	-	96.3	-	-	96.5	-

Table 3: The experimental results of each step for Case05 to Case07.

Bench.	Case05				Case06				Case07			
	|gp|	Cov.	|gp reduced|	CPU	|gp|	Cov.	|gp reduced|	CPU	|gp|	Cov.	|gp reduced|	CPU
Initial	3098	-	-	-	5008	-	-	-	4352	-	-	-
Fault Collapsing	2810	26.9	288	<0.001	4628	17.1	380	<0.001	3969	12.3	383	<0.001
MA Calculation	2275	77.0	535	0.17	1447	89.3	3181	2.42	1820	81.3	2149	11.83
Structural Matching	2029	100	246	<0.001	1132	97.1	315	0.01	1492	91.8	412	<0.001
Red. Fault Identification	2029	100	0	9	1132	97.1	0	2	1480	92.3	12	4
Total 1	-	-	-	9.17	-	-	-	4.43	-	-	-	15.83
100%	2029	100	0	1211.81	1020	100	112	3803.51	1238	100	170	3995.30
Total 2	-	-	-	1221	-	-	-	3808	-	-	-	4011

Case08, to evaluate the performance of each team. However, for confidentiality reasons, the Case08 benchmark was not publicly released from the Contest such that we cannot get the benchmark for the experiments.

The experimental results of our work and the top three teams in the Contest are summarized in TABLE 2. In TABLE 2, the first four columns show the benchmark information, which are all combinational. Columns 5 and 6 show the numbers of initial fault groups (Init.) and the golden results (Gold.). The golden results represent the maximal grouping of equivalent faults, which were obtained from the Contest. Any two fault groups in the golden result can be distinguished by at least one input vector. The next columns show the announced results of the top three teams, including the number of equivalent fault groups (|gp|), the CPU time measured in second (CPU), and the coverage (Cov.), from the Contest. The Cov. column is calculated by (Init. - |gp|) / (Init. - Gold.). Column Ours shows the corresponding results of our approach. We also show the CPU time of an approach for having 100% coverage in the last column. The approach for having 100% coverage uses SAT solvers to verify if any two fault groups from our results are functionally equivalent after conducting all the steps in Section III.

TABLE 3 also breaks down the results. The Cov. column represents the accumulated coverage from the initial step to the current step. The |gp reduced| column shows the number of fault group reduced in each step, and the CPU column represents the CPU time in each step. The Total 1 row shows the total CPU time used for all the steps in Section III. The Total 2 row shows the amount of CPU time needed for achieving 100% coverage. Our approach cost 9 seconds in the redundant fault identification step without reducing any group number for Case05. For Case07, the redundant fault identification step merges 12 fault groups while cost 4 seconds. However, for achieving 100% coverage, additional 3995 seconds were needed.

According to TABLE 2, our work achieved higher or equal coverages for all the benchmarks than the top three teams. Our average coverage is 96.5%. Furthermore, we can observe from TABLE 3 that the first three steps of our approach can achieve high coverages efficiently for each case. In some cases, we may spend extra time performing the step 4 without getting additional coverages. Nevertheless, with the step 4, we can identify redundant faults in acceptable CPU time.

5 CONCLUSION

Injecting artificial faults can examine whether the verification environment differentiates the faulty and fault-free designs. By maximally identifying equivalent faults, we can improve the efficiency of verification environment qualification. In this work, we propose a hybrid approach to identify equivalent faults in a circuit. In the experiment, the results show that our approach reached the highest average coverage on the hidden benchmarks compared to the winners of the Contest within 20 seconds CPU time.

REFERENCES

[1] A. Benso *et al.* 2007. A functional verification based fault injection environment. In *Proc. Defect and Fault-Tolerance in VLSI Systems.*
[2] A. Fin *et al.* 2000. A VHDL error simulator for functional test generation. In *Proc. Design, Automation and Test in Europe.*
[3] A. Veneris *et al.* 2002. Design rewiring using ATPG. In *IEEE Trans. CAD.*
[4] F. Ferrandi *et al.* 1998. Implicit test generation for behavioral VHDL models. In *Proc. Int. Test Conference.*
[5] F. V. Andrade *et al.* 2008. Improving SAT-based combinational equivalence checking through circuit preprocessing. In *Proc. of Int. Conference on CAD.*
[6] H. Y. Lin *et al.* 2012. A probabilistic analysis method for functional qualification under mutation analysis. In *Proc. Design, Automation and Test in Europe.*
[7] J. Arlat *et al.* 1993. Fault injection and dependability evaluation of fault-tolerant systems. In *IEEE Trans. Computers.*
[8] J. C. Miller *et al.* 1963. Systematic mistake analysis of digital computer programs. In *Commun. ACM.*
[9] L. Pintard *et al.* 2013. Fault injection in the automotive standard ISO 26262: An initial approach. In *Proc. European Workshop on Dependable Computing.*
[10] L. Pintard *et al.* 2014. From safety analyses to experi-mental validation of automotive embedded systems. In *Proc. of Pacific Rim International Symposium on Dependable Computing.*
[11] L. Pintard *et al.* 2015. Using fault injection to verify an AUTOSAR application according to the ISO 26262. In *Society of Automotive Engineers Technical Paper.*
[12] M. Hampton *et al.* 2007. Leveraging a commercial mutation analysis tool for research. In *Proc. Testing: Academic and Industrial Conference Practice and Research Techniques - Mutation.*
[13] S. Tasiran *et al.* 2001. Coverage metrics for functional validation of hardware designs. In *IEEE Design and Test of Computers.*
[14] W. Kunz *et al.* 1994. Recursive learning: A new implication technique for efficient solutions to CAD problems-test, verification, and optimization. In *IEEE Trans. CAD.*
[15] W. Kunz *et al.* 2005. Hardware design verification: Simulation and formal method-based approaches. In *Prentice Hall.*
[16] International Organization for Standardization. [n. d.]. https://www.iso.org/.
[17] R. Hamlet. 1977. Testing programs with the aid of a compiler. In *IEEE Trans. Software Engineering.*
[18] ICCAD. [n. d.]. http://cad-contest-2016.el.cycu.edu.tw/CAD-contest-at-ICCAD2016/.
[19] Synopsys. [n. d.]. https://www.synopsys.com/.