Error Correction Based on Verification Techniques

Shi-Yu Huang
Dept. of Electrical & Computer Engr.
U. of California, Santa Barbara
Santa Barbara, CA 93106

Kuang-Chien Chen
Fujitsu Labs of America
3350 Scott Blvd. Bldg. 34
Santa Clara, CA 95054

Kwang-Ting Cheng
Dept. of Electrical & Computer Engr.
U. of California, Santa Barbara
Santa Barbara, CA 93106

abstract

In this paper, we address the problem of correcting a combinational circuit that is an incorrect implementation of a given specification. Most existing error-correction approaches can only handle circuits with certain types of errors. Here, we propose a general approach that can correct a circuit with multiple errors without assuming any error model. We identify internal equivalent pairs to narrow down the possible error locations using local BDD's with dynamic support. We also employ a technique called back-substitution to correct the circuit incrementally. This approach can also be used to verify circuit equivalence. The experimental results of correcting fully SIS-optimized benchmark circuits with a number of injected errors will be presented.

1. Introduction

During the design cycle, the correction of a circuit which has been proven different than a given specification is an essential but difficult task. Recently, the efficiency of gate level equivalence checking tools that can check circuit equivalence has dramatically improved [5,10]. However, error correction is still a challenging problem. In general *error diagnosis* is performed first to serve as the guidance of rectifying the circuit. Existing error diagnosis approaches can be divided into two categories: (1) OBDD-based approaches [2,6,9], and (2) simulation-based approaches [3,8]. For the OBDD-based approaches, one or two candidate signals which can correct the entire circuit are identified using BDD formulas. This approach will fail if no such candidate signal exists. In addition, it suffers from the memory explosion problem for large designs. For simulation-based approaches, input vectors that differentiate the erroneous circuit from its specification are simulated to narrow down the possible error regions. This approach is more flexible because it can still find a set of suspected incorrect signals for a circuit with multiple errors. However, this approach is not as precise as the OBDD-based approach, and thus, it provides less information about how to rectify the circuit after diagnosis.

For the error correction step, earlier research used a classification of commonly occurred design errors [1]: (1) gate-type error, including a missing gate, an extra gate and wrong gate type (2) connection-type error, including a missing connection and an extra connection. Using this error model and the information provided by error diagnosis, they tried to match the error with one type of error in the error model. If an error does not belong to any type in the model, then no solution can be produced.

Engineering change (EC) is a problem closely related to error-correction. Its goal is to reuse the existing investment on the implementation of a circuit when a specification is slightly changed. An EC algorithm was proposed in [7] assuming three netlists are given: the new specification, the old specification and the old implementation. This approach uses automatic test pattern generation (ATPG) techniques to identify equivalent signal pairs between the new specification and the old implementation, for the purpose of reusing some existing logic gates. This identification process proceeds from the input side towards the output side in stages. Once an equivalent pair is found, the signal in the new specification can be replaced by its counterpart in the old implementation. Also, a technique called *back-substitution* is employed to create more equivalent signal pairs using structural correspondence between the old specification and the new specification. The function of the old implementation is changed when back-substitution is performed, and it is incrementally transformed to a function equivalent to the new specification.

In this work we borrowed the concept of engineering change and developed new techniques to correct a circuit with multiple errors. For error diagnosis, we used a novel technique called *dynamic support* for building BDD's to identify equivalent signal pairs in stages. Also, we used a dominator as a pseudo-output to take the observability don't cares into consideration. This technique is superior to the ATPG-based approach when a specification is optimized using a lot of don't cares, and thus, has less structural similarity to the original specification. After the identification of the equivalent signal pairs, we incorporate a new method to locate the error signals. Then the back-substitution is employed to correct the identified error signal incrementally.

The rest of this paper is organized as follows. In Section 2, we introduce the terminologies and the overall scenario. In Section 3, we discuss the technique of dynamic support and the error correcting process in detail. In Section 4, we present experimental results. Section 5 gives concluding remarks.

2. Preliminaries

2.1 Definitions

Without loss of generality, we assume that the specification and the incorrect implementation are both single-output circuits. Both circuits are given as gate level netlists and denoted as C_1 and C_2, respectively. Our algorithm is performed on a netlist where the primary inputs of 2 netlists are joined together. The outputs for C_1 and C_2 are denoted as o_1 and o_2. The problem of error correction is to find a transformation that can convert C_2 to a circuit equivalent to C_1. During this transformation, the output function of C_1

* This work was supported by the National Science Foundation under grant MIP-9503651, California MICRO and Fujitsu Labs of America.

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

remains the same, while the internal signals of C_2 could be changed to rectify the function of o_2.

Definition 1: (Signal pair): (a_1, a_2) is called a signal pair, where a_1 is a signal of C_1, and a_2 is a signal of C_2.

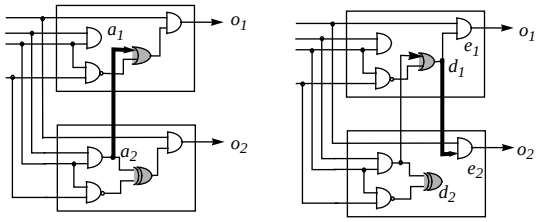
Definition 2: (Equivalent pair): (a_1, a_2) is called an equivalent pair if the values of signal a_1 and a_2 in response to any input vector are identical.

Definition 3: (Discrepancy function): A vector v is a distinguishing vector for a signal pair (a_1, a_2) if the application of v can produce (0, 1) or (1, 0) at a_1 and a_2. The characteristic function of the set of distinguishing vectors is called a *discrepancy function* and denoted as $Disc(a_1, a_2)$.

Definition 4: (Generalized equivalent pair): (a_1, a_2) is called a generalized equivalent pair with respect to o_1 if the function of o_1 remains the same after signal a_1 is replaced by a_2. This generalized definition is implicitly used in the sequel.

Definition 5: **(**Back-substitution): An operation that replaces a signal a_2 in C_2 by a signal a_1 in C_1 is called a back-substitution. Note that (a_1, a_2) is usually not an equivalent pair to be considered for back-substitution.

Note that back-substitution does not preserve the output function of C_2. It is an operation that tries to create more equivalent pairs between the specification C_1 and the incorrect implementation C_2 after an error signal is identified, so that more subsequent normal substitutions can be performed. The selection of the best candidate for back-substitution may affect the result. Fig. 1 illustrates the difference between normal substitution and back-substitution. In this example, the incorrect circuit is different



(a) replacing a_1 with a_2 (normal substitution)

(b) d_2 is back-substituted by d_2 (function of o_2 is changed)

Fig. 1: Normal substitution and back-substitution.

from the specification by only one gate (shadowed). Signal pair (a_1, a_2) is equivalent; hence a normal substitution is performed. Signal pair (d_1, d_2) is not equivalent and they are selected for back-substitution to create more equivalent pairs in their fanout-cones. After this back-substitution, o_2 becomes equivalent to o_1 and one more gate (e_2) in C_2 can be reused to implement a correct circuit.

2.2 The overall procedure

Given a specification C_1 and an implementation C_2, our algorithm first constructs the joined netlist. Then we compare the signal names in the two circuits to pair up signals. If a signal a_1 in C_1 has a corresponding signal a_2 with the same name in C_2, then signal a_1, a_2 are regarded as *key signals* and (a_1, a_2) is a *key signal pair*. The joined netlist can be transformed into a hypergraph consisting of a set of supernodes and a set of connections such that every connection is a key signal and every supernode corresponds to a collapsed subcircuit in the original netlist. We sort the key signals in an order that every signal is after its transitive fanins. Once an equivalent signal pair (a_1, a_2) is identified, a_1 is replaced by a_2 immediately. This incremental approach identifies equivalent pairs in stages to significantly reduce the run-time complexity [7]. After we have checked every internal key signal pair, we check if the output functions o_1 and o_2 are equivalent. If they are equivalent, then no more correction is needed. Otherwise, we select an inequivalent pair and perform back-substitution. After the back-substitution, another iteration of error-diagnosis is followed to identify newly generated equivalent pairs. The process continues until o_1 and o_2 are equivalent. The overall procedure of this incremental error correction process is described in Fig. 2.

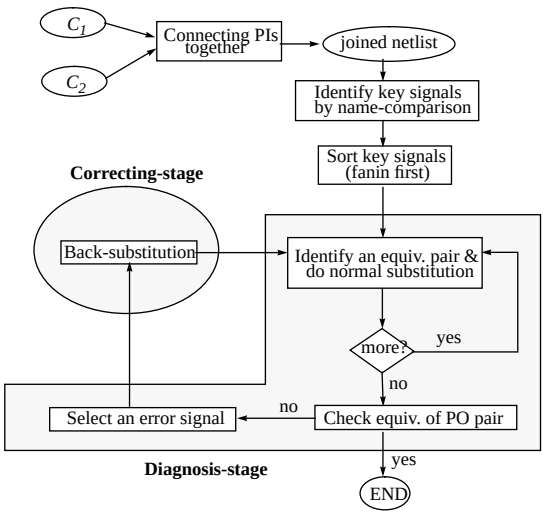


Fig. 2: Overview of our incremental error-correcting process.

3. Techniques

3.1 Dynamic-support for constructing BDD's

Here we discuss a technique to check the equivalence of a signal pair using dynamic supports for constructing BDD's. We first define the sensitization function.

Definition 6: Sensitization function is the characteristic function of the set of input vectors that can propagate a discrepancy signal from signal a_1 to a primary output o_1, denoted as $Sen(a_1, o_1)$. This function is a boolean difference of function o_1 with respect to an internal signal a_1, i.e., $Sen(a_1, o_1) \equiv (o_1(a_1{=}0)) \oplus (o_1(a_1{=}1))$.

Property 1 [5,6]: (a_1, a_2) is an equivalent pair if the intersection of the discrepancy function and the sensitization function is the zero function, i.e., $Disc(a_1, a_2) \wedge Sen(a_1, o_1) = \underline{0}$. ❑

This property states that if there exists no input vector that can generate (0, 1) or (1,0) at (a_1, a_2) and propagate this discrepancy all the way to the primary output o_1, then it is safe to replace signal a_1 with signal a_2 without changing the function of o_1. If the global BDD of the joined netlist can be constructed, the above necessary and sufficient condition can be checked efficiently. However, for larger designs, it is not feasible to construct the global BDD. In [5], an ATPG technique is applied to check the necessary and sufficient condition of equivalence. The efficiency of this approach strongly relies on the degree of structural similar-

ity between the circuits. In this paper we further propose a new technique that is less sensitive to structural similarity to identify the equivalent signal pairs.

This technique is based on an observation that if there exists a cutset λ for the input-cones of a_1 and a_2 such that no value combination of the cutset can produce (0, 1) or (0, 1) at (a_1, a_2), then (a_1, a_2) is an equivalent pair. We further enhance this sufficient condition by considering the a_1's dominator, *dom*, (if it exists) as the pseudo primary output. Dominator is a signal in the output-cone of a_1, such that every path from a_1 to any primary output should passes through it. We denote the discrepancy function with respect to the cutset λ for signal pair (a_1, a_2) as $Disc_\lambda(a_1, a_2)$, and the sensitization function with respect to one of signal *a*'s dominator, *dom*, as $Sen_\lambda(a_1, dom)$. In the sequel, cutset is also referred to as *support*. The following lemma shows a sufficient condition of equivalence.

Lemma 1: A signal pair (a_1, a_2) is equivalent if there exists a support λ for the input-cones of a_1 and a_2 such that: $Disc_\lambda(a_1, a_2) \wedge Sen_\lambda(a_1, dom) = \underline{0}$, where *dom* is a dominator of signal a_1.

Our experiments show that a large percentage of equivalent pairs satisfy the above sufficient condition without using the primary inputs as the support. Hence we developed a heuristic that expands the support towards the primary inputs dynamically to select an appropriate support for checking the equivalence of a candidate pair. Since we only construct the discrepancy function and the sensitization function (if a dominator exists) based on a local support, we can handle much larger designs than the traditional approach using global BDD's. Fig. 3 illustrates this backward expansion process for selecting dynamic support.

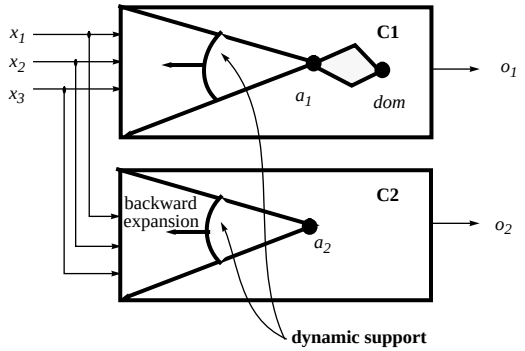


Fig. 3: The dynamic support that expands towards the primary inputs for verifying the equivalence of (a_1, a_2).

In the worst case, we need to advance the frontier of the support all the way to the primary inputs to prove whether a signal pair is equivalent or not, which requires the construction of global BDD's. To avoid this situation, we set a limit on the maximum levels of the backward expansion process. If the signal pair cannot be proven equivalent after reaching the limit, we then give up, treat them as inequivalent and move on to the next signal pair. The only exception is when the target signal pair is a primary output pair. Then we switch to the ATPG technique to continue the search for any possible distinguishing vector. The complete procedure of checking the equivalence for a signal pair is shown in Fig. 4.

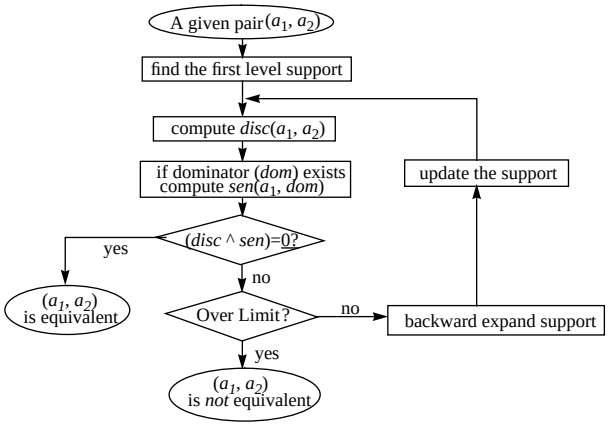


Fig. 4: Checking an equivalent pair using dynamic support.

3.2 Error signal selection

Once the process of identifying equivalent pairs is completed, a heuristic is applied to predict the possible locations of the error signals. In the joined netlist, we search for a cutset from the primary output towards the primary inputs. In this cutset, every signal *s* is a key signal, and has been identified as equivalent to its corresponding signal *s'*. Finding such a cutset is helpful for precisely locating the sources of errors. Fig. 5 illustrates the idea.

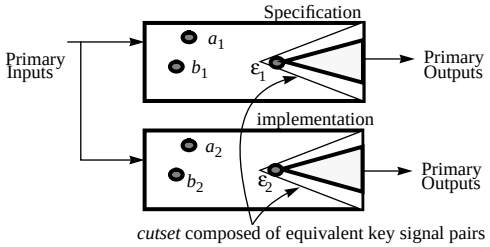


Fig. 5: Illustrating the cutset for selecting an error signal.

The key signals that were identified as inequivalent are in shadow. (a_1, a_2), (b_1, b_2) and (ε_1, ε_2) are three of them. A cutset that envelops signal pair (ε_1, ε_2) and their fanout cones is identified. Since the signal pairs (a_1, a_2) and (b_1, b_2) do not cause the key signal pairs in their output cones to become functionally different, their inequivalence is not responsible for the inequivalence of the primary output functions. On the other hand, the output cone of signal ε_2 is affected because of the incorrect function of signal ε_2, which leads to the incorrectness of the implementation's primary output functions. Therefore, signal ε_2 is the suspected error signal. We examine each key signal from the primary outputs towards the primary inputs to search for such an error signal for the subsequent correcting process. The candidate error signal satisfies the condition that *all of its fanin signals in the hypergraph are also in the identified envelop cutset*. Usually this signal is one of the sources of errors. Once an error signal is located, we perform the back-substitution to fix it.

4. Experimental Results

We have implemented this algorithm on top of SIS [4]. Note that our program can also be used for verifying circuit equivalence. We have used our program to verify the equivalence of three types of circuits synthesized by SIS: redundancy removed circuits, circuits minimized by *script.rugged*, and mapped circuits using library *synch.genlib*. The results of verifying minimized and mapped ISCAS85 benchmark circuits on a Sun Sparc-20 workstation equipped with 128MB memory are shown in Table 1 and 2, respectively. The results of verifying the redundancy removed circuits are faster than those in Table 1 and 2. But due to the space limitation, they are not presented here. Redundancy removed circuits are typically structurally similar to their original circuits. However, for thoroughly optimized (by *script.rugged*) or mapped circuits, the structural similarity has been significantly reduced, and thus, the verification is more difficult. Our approach is quite robust and applicable to circuits with different degrees of structural similarity. Our experience shows that it is very common that some equivalent signal pairs cannot be proven equivalent unless we backward expand the dynamic support for a number of levels. Currently we expand the support for up to 6 levels if necessary. If this is not enough to prove a signal pair equivalent, we regard them as inequivalent. Our experience shows that a very high percentage of equivalent pairs (the column of *# equiv. pairs* divided by the column of *# key pairs*) can be identified using this sufficient condition. We also implemented the idea of using a dominator as the pseudo-output. For some cases, this feature increases the number of equivalent pairs. Since we do not build global BDDs, the memory explosion problem has not been encountered.

Table 3 shows the results of correcting the minimized circuits with multiple errors (randomly injected 3 errors) [1]. The column labeled *reuse-rate* is the ratio of the number of nodes taken from the incorrect implementation to the total number of nodes in the final correct circuit. i.e., *reuse-rate* = (*no. of existing nodes*) / (*no. of existing nodes + no. of new nodes*).

Table 1: Verifying circuits minimized by *script.rugged*.

Circuit-name	C1 #node	C2 #nodes	# key signals	# eq. pairs	Time (sec)
C432	123	49	38	37	2.3
C499	162	162	90	90	1.0
C880	311	49	49	49	1.7
C1355	474	162	90	90	1.4
C1908	441	152	98	94	20.6
C2670	788	256	208	208	76.4
C3540	956	240	146	146	492.4
C5315	1467	401	292	292	13.1
C6288	2353	934	704	704	24.1
C7552	2165	578	398	398	47.4

Table 2: Verifying mapped circuits using library *synch.genlib*.

Circuit-name	C1 #node	C2 #nodes	# key signals	# equiv. pairs	Time (sec)
C432	123	99	8	8	18.8
C499	162	177	132	132	1.7
C880	311	163	76	76	1.5
C1355	474	445	36	36	16.9
C1908	441	436	436	425	7.6
C2670	788	518	211	211	91.0
C3540	956	632	160	160	72.1
C5315	1467	952	307	307	12.6
C6288	2353	2309	61	61	105.8
C7552	2165	1289	458	458	42.4

Table 3: Correcting minimized circuits injected with 3 random errors.

Ckt-name	# key signals	#equiv. pairs	# new nodes	#exist. nodes	Reuse rate	Time (sec)
C432	38	36	16	187	92%	8.0
C499	90	88	4	437	99%	6.7
C880	49	45	40	327	89%	12.1
C1355	90	89	6	447	99%	10.6
C1908	98	94	79	369	82%	31.4
C2670	208	205	27	918	97%	88.9
C3540	145	136	147	1167	89%	2025.0
C5315	292	283	105	1373	93%	27.0
C6288	704	701	14	2362	99%	35.8
C7552	398	395	190	2261	92%	42.4

5. Conclusions

Most existing algorithms for automatic error correction are restricted to certain types of errors or cannot handle large designs with multiple errors. In this paper, we present a general approach for rectifying a large combinational circuit. Since no error model is assumed, this approach can handle arbitrary types of errors. In the process of error diagnosis, we proposed a dynamic-support technique to identify the equivalent pairs efficiently using local BDD's. Based on the information derived in the error diagnosis phase, we narrow down the possible locations of errors for the subsequent correcting process. To correct those identified error signals, we perform a sequence of back-substitutions to rectify the circuit incrementally. This approach is less sensitive to the structural similarity than the previous incremental approaches. We have successfully verified all fully SIS-optimized ISCAS85 benchmark circuits. Also, we have successfully corrected the circuits when injected with multiple random errors.

References

[1] K. A. Tamura, "Locating Functional Errors in Logic Circuits," ACM/IEEE, *Design Automation Conference*, pp. 185-191, 1989.

[2] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the Diagnosis and the Rectification of the Design Errors with PRIAM," Proceedings of *ICCAD*, pp. 30-33, 1989.

[3] M. Tomita, H. H. Jiang, T. Tomamoto, and Y. Hayashi, "An Algorithm for Locating Logic Design Errors," Proceedings of *ICCAD*, pp. 468-471, 1990.

[4] "SIS: A System for Sequential Circuit Synthesis," Report M92/41, University of California, Berkeley, 1992.

[5] D. Brand, "Verification of Large Synthesized Designs," Proceedings of *ICCAD*, pp. 534-537, 1993.

[6] P. Y. Chung, Y. M., Wang, and I. N., Hajj, "Diagnosis and Correction of Logic Design Errors in Digital Circuits," *ACM/IEEE Design Automation Conference*, pp. 503-508, 1993.

[7] D. Brand, A. Drumm, S. Kundu, and P. Narrain, "Incremental Synthesis," Proceedings of *ICCAD*, pp. 14-18, 1994.

[8] A. Kuehlmann, D.I. Cheng, A. Srinivasan, and D.P. LaPotin, "Error Diagnosis for Transistor-level verification," ACM/IEEE *Design Automation Conference*, pp. 218-223, 1994.

[9] C. C. Lin, K. C. Chen, S. C. Chang, M. Marek-Sadowska, and K.T. Cheng, "Logic Synthesis for Engineering Change," ACM/IEEE *Design Automation Conference*, pp. 647-652, 1995.

[10] S. M. Reddy, W. Kunz, and D. K. Pradhan, "Novel Verification Framework Combining Structural and OBDD Methods in a Synthesis Environment," ACM/IEEE *Design Automation Conference*, pp. 414-419, 1995.