Design Error Diagnosis and Correction in VLSI Digital Circuits *

Andreas G.Veneris
University of Illinois at
Urbana-Champaign
Coordinated Science Laboratory and
Department of Computer Science
e-mail: aveneris@students.uiuc.edu

Ibrahim N.Hajj
University of Illinois at
Urbana-Champaign
Coordinated Science Laboratory and
Department of Electrical and
Computer Engineering
e-mail: hajj@uivlsi.csl.uiuc.edu

Abstract

With the increase in the complexity of VLSI circuit design and the corresponding increase in the number of logic gates on a chip, logic design errors can frequently occur. In this paper we present an efficient test–vector simulation approach for design error diagnosis when a small number of modifications can rectify the erroneous design. We also compare the quality of verification and diagnosis that test vector simulation offers for design errors with the one offered by BDDs and show the competitive performance of the former. Experimental results exhibit the robustness of our approach.

1 Introduction

Logic design errors may occur during the design cycle of a VLSI chip when the designer has to adjust manually the netlists generated by synthesis tools in order to satisfy different design goals. In this paper we present an efficient test vector simulation approach for combinational circuit Design Error Detection and Correction (DEDC) when a small number of modifications on a single line are sufficient to correct an erroneous design. We also compare the quality of single design error verification and diagnosis offered by test simulation with the one offered by Boolean function manipulation and show the competitive performance of the former.

Our algorithm is based on a single source logic design error model, an *extension* of the one proposed in [1] (Fig.1). An experimental study described in [2] has shown that single design errors as defined in [1] cover 97.8% of the design errors that can occur during a real life manual resynthesis procedure. In the verification phase of our algorithm, we apply test vectors generated for detecting stuck-at faults [7] and, if necessary, vectors generated by a random test vector generator. On a theoretical basis, Abadir et al [1] proved that a complete set of test–vectors for stuck-at faults detects the majority of the design errors of their model and has a very good chance of detecting the remaining ones. This result of [1] was experimentally confirmed in [3]. In our experiments, unless the error injected is *redundant*, that is, it does not change the functionality of the design [5], the test vector set we use always contains vectors that excite and propagate the error to some primary output(s). Consequently, if the response at the primary outputs of the gate–level description is correct for all test–vectors then we can conclude with high confidence that the design is correct; otherwise we keep the vectors with incorrect primary output responses. These vectors provide information on the set and nature of possible candidate modifications during the execution of the algorithm.

In [1], Abadir et al presented a design error model (Fig.1) which has been adopted by most of the methods for the DEDC problem. These methods can be divided into two main categories with respect to the underlying technique used; the ones based on Boolean function manipulation techniques [5], [6], [10] and the ones based on test vector simulation [8], [9]. In [5] an error equation guides both the error location and error correction procedures of the diagnosis algorithm. The work in [10] uses for correction the results of an error equation as well, but error location is performed with a procedure similar to the one presented in this paper. In the work of Huang et al [6] the memory explosion problem of the BDDs is eliminated but the method relies on the existence of the ideal gate–level implementation. The work in [9] is based on simulation of vectors that produce erroneous primary output responses but the generation of these vectors is done with the use of BDDs which can limit the applicability of the method. Finally, [8] uses multiple gate type changes only and presents a test vector simulation approach that does not guarantee to return a solution even if there exists one.

We can conclude from the above discussion that most of the methods that consider the DEDC problem are based on Boolean function manipulation procedures. Although there exist methods based on BDDs that guarantee to return a solution such as [5], [6], and [10], if such a solution exists, these methods may not be applicable to some

*This research was supported by the Joint Services Electronics Program under Contract #N00014-96-1-0129.

circuits due to the exponential memory requirements of the underlying data–structures. This is not the case for a method based on test simulation which is always applicable to large circuits. Moreover, our algorithm guarantees to return a solution, if such a solution exists in our modification model, provided that the set of test vectors that activate the inconsistencies is non-empty. In addition, the run time results of the approach described in this paper are orders of magnitude better than previous methods for simple design error diagnosis and the proposed method does not require any information about the correct gate–level description of the circuit.

In the next section we give the formal definitions and theoretical basis of our diagnosis algorithm. Section 3 describes the error detection and error correction procedures, section 4 presents the experimental results and the last section contains the conclusions and future directions.

2 Problem Formulation

The *inputs* to the algorithm are the *functional specification* F_C of a circuit C that describes the correct behavior of C, an incorrect *gate–level description* G_C of C, and a set of vectors V_{act} where each member of the set produces a different response at one or more primary outputs of F_C and G_C. V_{act} is compiled during the initial circuit verification step of the algorithm from vectors that detect stuck-at faults [7] and, if necessary, vectors from random test vector generation. During our experiments, unless the injected error is redundant, V_{act} is never empty. The *output* of the algorithm is a list of the *actual* and all *equivalent* modifications from our design modification model that rectify the circuit. The existence of equivalent modifications follows from the fact that there can be more than one way to synthesize a function and rectify the design [5]. We now present our modification model which, if we exclude type j (Fig.1), it is an extension of the one presented in [1].

Definition 1 *With respect to Fig.1, let a* **modification** *be either a* **Wrong Gate***, namely types a, b, c, e, f, i, or a* **Misplaced Wire***, namely types d, g, h, or a* **Wrong Gate/Misplaced Wire***, an occurrence of both errors on a single gate. We say that an incorrect gate–level description G_C is* **single source correctable** *if there exists a line l in C and one modification on the gate driving l that rectifies G_C.*

In the discussion that follows, we examine incorrect combinational gate–level descriptions with `NOT`, `AND`, `NAND`, `OR` and `NOR` gates. We introduce one `BUFFER` for every fan-out line of a branch. We say that a line l, fan–in to an `AND` or `NAND` (`OR` or `NOR`) gate has *controlling value* for vector v if the value of l is 0 (1) when v is simulated. If l drives a `NOT` or a `BUFFER` the value of l is always a controlling one. In the following definition, we borrow from the concept of Error Possibility Index [9].

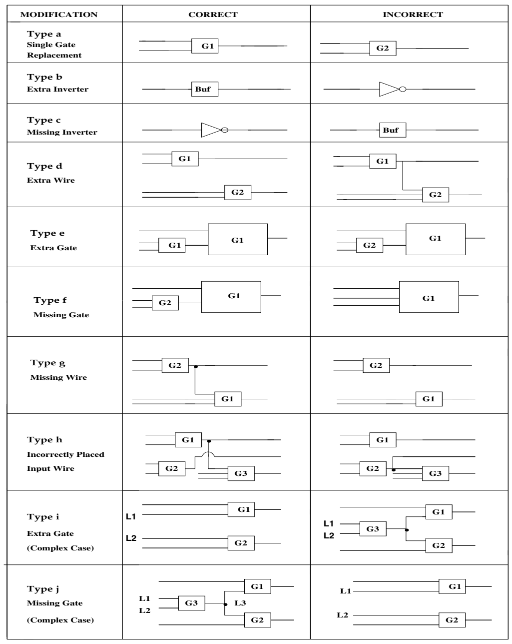

Figure 1: Abadir et al Design Error Model

Definition 2 *Let $v \in V_{act}$ and let primary output PO_i with* incorrect *response under v. We define the* **observability measure,** *OM_l^i, of a line l for v and over PO_i recursively as follows:*

(a) If l is a primary output then $OM_l^i = 1$ if $l = PO_i$, and $OM_l^i = 0$, otherwise.

(b) if l is fan-in of a gate G and l' is the fan-out of G then:

- *$OM_l^i = 0$, if l has non-controlling value and some other fan-in of G has controlling value.*
- *$OM_l^i = OM_{l'}^i$, if l has non-controlling value and all fan-ins of G have non-controlling value.*
- *$OM_l^i = \frac{OM_{l'}^i}{cv}$, if l has controlling value, where cv is the number of fan-ins of G with controlling value.*

(c) if l is a fan-out stem, then $OM_l^i = min(1, \sum_{l'} OM_{l'}^i)$, l' drives a buffer at the fan-out branch of l.

Intuitively, the observability measure OM_l^i of a line l over an erroneous primary output PO_i for $v \in V_{act}$ denotes the *potential* of l to change PO_i into its correct value (for v). Its recursive definition makes its computation efficient. The theorem that follows implies the correctness of our approach which is presented in section 3.1.

Theorem 1 *Let l be an actual or equivalent modification location for a single source correctable G_C. If $v \in V_{act}$ and PO_i has erroneous response for v then $OM_l^i = 1$ (for v).*

Proof: The proof of the theorem is by induction. Let a *k-cut* on C be the set of all lines of C at level k, where the level of the primary outputs is 0, their immediate fan-ins 1 etc and the level of a fan-out stem is equal to the maximum level of its branches plus one. Let j be the level of l, and let MS_i^L be any minimal set of lines l' of the i-cut, $i \leq j$, such that when their values are complemented they correct all erroneous primary outputs for v. We claim that $\sum_{l' \in MS_i^L} OM_{l'}^i \geq 1$. Since, by definition, the value OM_l^i cannot exceed 1, proving the claim the theorem follows.

To prove the claim, we use induction on the level of cuts starting from PO_i towards l. For the base case, $OM_{PO_i}^i = 1$ (Definition 2(a)) and the claim holds. Let it be true for all such minimal sets including the ones of the k-cut. To prove that it holds for all minimal sets of the $(k+1)$-cut, let MS_{k+1}^L be any such set. Let S_k be the set of lines of the k-cut compiled as follows: for every set of lines of MS_{k+1}^L that drive the same gate G, S_k contains the output of G and for every line $l' \in MS_{k+1}^L$ that drives a branch, S_k contains *all* fan-out branches of l'. It is straightforward to see that S_k contains at least one minimal set of lines MS_k^L (otherwise MS_{k+1}^L is not minimal). Moreover, if l' is a line of MS_k^L driven by gate G, then by Definition 2(b) the minimal set of lines fan-in to G that belong in MS_{k+1}^L should have a sum of observability measures equal to $OM_{l'}^i$. If they don't, they will not be able to change the value of l'. If l'' is a fan-out stem of MS_{k+1}^L, then by Definition 2(c) l'' will have an observability measure value equal to the sum of the observability measure values of *all* of its branches (or 1). By pigeonhole principle, the sum of the observability measures of all lines of MS_{k+1}^L should be greater or equal to this of the MS_k^L ones. This proves the claim and completes the proof.

The significance of Theorem 1 lies in the fact that it provides a tool to screen out candidate modification locations that do not satisfy the condition of the theorem and, therefore, cannot be an actual or equivalent modification location. Observe that the theorem is one–way only. A line l might satisfy the condition of the theorem and not be an actual or equivalent modification location.

3 The Algorithm

3.1 Error Detection

The first stage of the Error Detection algorithm simulates vectors of V_{act} (parallel fault simulation) and computes the observability measure values of the lines of the circuit. Next, it applies Theorem 1 iteratively so that lines that do not qualify get deleted from the list of potential modification locations, C_{error}. This is repeated for all vectors from V_{act} so that the size of C_{error} is reduced significantly. Initially, C_{error} contains all the lines of the circuit. Due to the recursive definition of the observability measure, its evaluation takes place in a backward fashion, that is, from primary outputs towards the primary inputs.

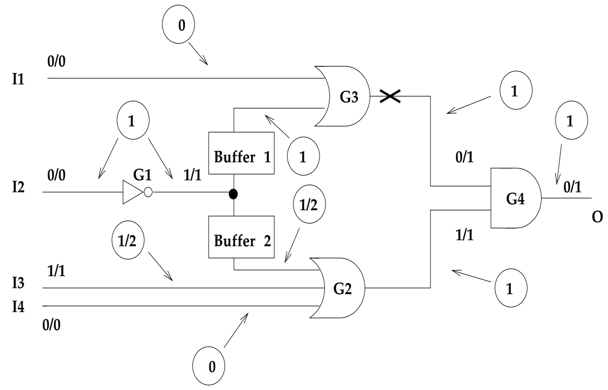

Figure 2: Erroneous circuit for Example 1

Example 1: Fig.2 shows the *incorrect* implementation of a circuit C and the values of all lines of C when we apply input vector $V = (I_1, I_2, I_3, I_4) = (0, 0, 1, 0)$. The *correct* implementation of C is obtained when G_3 is replaced by an `AND` gate. Vector V activates this difference and the tuples on the lines are pairs of correct/incorrect situation values. The observability measure values of the lines are in circles and are computed from the primary output O towards the primary inputs $I1$, $I2$, $I3$, $I4$. Observe that $OM_{G_1}^O = 1$, but no single modification, from our model, applies on G_1 and corrects the design. Location G_1 is screened out from the set of candidate locations when Theorem 1 is applied for input vector $V' = (1, 1, 1, 0)$. In this case $OM_{G_1}^0$ is 0 and G_1 gets deleted from C_{error}.

The second, and last, step of the error location procedure is *Inverted Simulation.* During the verification stage of our algorithm, we create a linked–list of bit values at every line of the circuit, namely the *faulty bit–list.* The i-th bit of this list for line l holds the value of l when the i-th vector that activates the inconsistencies is simulated. During Inverted Simulation we complement the values of this faulty bit–list for every remaining candidate $l \in C_{error}$ separately and resimulate, in parallel, all bits of the faulty bit–lists at the fan-out cone of l. Since every actual or equivalent modification location l should now give a correct primary output response, candidate locations with incorrect primary output responses are deleted from C_{error}.

3.2 Error Correction

During error correction the algorithm exhaustively compiles a list of all possible corrections for every remaining candidate of C_{error}. In detail, for every candidate line $l \in C_{error}$ it applies all modifications of the modification model (Definition 1) , one at a time, on the gate driving l and keeps the correction if and only if it gives complemented faulty bit–list values at l. Once the above list of potential corrections has been compiled we perform an additional verification step for each potential correction of the list. For the purpose of this verification step we use the vectors for stuck-at faults that did not activate the inconsistencies in the first place, and vectors produced by a random test–vector generator. If a correction produces erroneous primary output responses for some vector then it is deleted from the list of potential corrections. The modifications that qualify at the end of this procedure are also the output of the algorithm.

4 Experimental Results

We implemented the above algorithm in C language, ran it on a Sparc 10 workstation and tested it on ISCAS'85 benchmark circuits for all three types of design errors of our model. The type of error and the location where it was injected every time was selected randomly. We repeated the experiment 30 times for each circuit and the average values of the results are reported in Table 1. All run–times are in seconds.

The first column of Table 1 contains the initial size of C_{error}, that is, all the lines of the circuit. The next column contains the average time needed for the initial circuit verification step. As explained in section 2, this step simulates vectors for stuck-at faults, and random vectors, if necessary. This run–time also includes the time required to compile V_{act}. In our experiments, V_{act} is never empty unless the error injected is redundant [5]. The average size of V_{act} required for an efficient solution to DEDC is given in the next column. Columns 4, 5 and 6 contain the average times of the error location, error correction and correction verification for *all* corrections, respectively. The correction verification time does not include the time needed for random test vector simulation. This time is proportional to the one shown in column 2. The worst case run–time behavior of the proposed approach was less than three times the average case. Column 7 contains the number of random test vectors used during the verification step for the proposed corrections and the next column has their hit–ratio to propagate the error to some primary output(s). The last column contains the correction hit–ratio; that is, the percentage of the corrections that rectify the design using formal verification techniques [4] versus the total number of corrections returned by the algorithm. The high hit–ratio indicates that diagnosis provided by test–vector simulation is a good and safe alternative to the one based on BDDs. As a final step in our experiments, we verified the corrections with a larger set of random test vectors than the one reported in column 7 but the correction hit–ratio increase was negligible.

5 Conclusions and Future Work

In this paper, we first defined a design error scenario and then gave a test vector simulation diagnosis algorithm that returns all possible actual and equivalent modification locations with their respective corrections. The experimental results show the robustness of the proposed method and suggest that test simulation is a good alternative to BDDs for verification and diagnosis of simple design errors. Our future work includes extenting the proposed approach to circuits with multiple design errors.

References

[1] M.S.Abadir, J.Ferguson, and T.E.Ferguson. "Logic Verification via Test Generation". *IEEE Trans. on CAD, vol.7, pp.138–148,* January 1988.

[2] E.J.Aas, K.Klingsheim, and T.Steen. "Quantifying design quality: A model and design experiments". *Proc. of EURO–ASIC, pp.172–177,* 1992.

[3] H.A.Asaad, and J.Hayes. "Design Verification via Simulation and Automatic Test Pattern Generation". *Proc. of IEEE ICCAD, pp.174–180,* 1995.

[4] R.E.Bryant. "Graph–Based Algorithms for Boolean Function Manipulation". *IEEE Trans. on Computers, vol.C–35, no.8, pp.677–691,* 1986.

[5] P.-Y.Chung, Y.-M.Wang, and I.Hajj. "Logic Design error diagnosis and correction". *IEEE Trans. on VLSI Systems, vol.2, pp.320–332,* September 1994.

[6] S.-Y.Huang, K.-C.Chen, and K.-T.Cheng. "Error Correction Based on Verification Techniques". *Proc. of ACM/IEEE DAC , pp.258–261,* 1996.

[7] T.M.Niermann, and J.H.Patel. "HITEC: A test generation package for sequential circuits". *Proc. of European Automation Conf., pp.214–218,* 1991.

[8] I.Pomeranz and S.M.Reddy. "On diagnosis and correction of design errors". *IEEE Trans. on CAD, vol.14, pp.255–264,* February 1995.

[9] M.Tomita, T. Yamamoto, F.Sumikawa, and K. Hirano. "Rectification of Multiple Logic Design Errors in Multiple Output Circuits". *Proc. of ACM/IEEE DAC, pp.212–217,* 1994.

[10] A.Veneris, and I.Hajj. "A Fast Algorithm for Locating and Correcting Simple Design Errors in VLSI Digital Circuits". *Proc. of 7th IEEE GLS–VLSI, pp.45–50,,* 1997.

ckt name	C_{error}	Verif. time	$\mid V_{act} \mid$	Err.Loc. Time	Err.Corr. Time	Corr.Verif. Time	# random vectors	rand.vect. hit–ratio	Corr. hit–ratio
C432	547	4.1	40	2.8	4.2	0.8	1200	10.8 %	97.8%
C499	1252	7.0	50	16.0	7.5	1.1	3500	21.7%	96.0%
C880	908	5.0	50	7.2	5.4	0.8	1200	18.1%	100%
C1355	1388	7.8	60	15.3	7.8	1.1	3000	20.0 %	99.7%
C1908	1930	8.2	70	16.4	10.5	1.7	4000	18.5 %	99.0%
C2670	2720	10.9	80	15.9	10.9	2.0	4200	17.3%	97.2%
C5315	5315	11.4	80	20.7	11.4	1.7	4500	24.9 %	92.5%
C7522	7660	15.3	110	22.3	14.9	2.1	5000	29.3 %	94.9%

Table 1