A Fast Algorithm for Locating and Correcting Simple Design Errors 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 approach to Design Error Detection and Correction when a small number of modifications can rectify the design. Our method is based on test vector simulation and Boolean function manipulation techniques. The proposed approach guarantees to return a solution, if such a solution exists in our modification model, in a short computational time. Experimental results show the robustness of our approach.

1 Introduction

During the design cycle of a VLSI chip it is common that the designer has to apply manually a few adjustments to the netlists generated by synthesis tools in order to satisfy certain design goals such as area, timing and power consumption optimization constraints. This manual resynthesis procedure might introduce some design errors and the task of the designer is to verify the correctness of the new implementation and correct it if the resynthesis procedure has produced an erroneous design. Since logic verification tools ([1], [3]) are not able to provide information about the location and nature of the desired corrections it is necessary to develop automatic methods that perform Design Error Detection and Correction (DEDC) effectively. In this paper we present a new approach to the DEDC problem for *combinational circuits* when a small number of modifications on a single line are sufficient to correct the design.

This work provides a solution to the DEDC problem in a short computational time. We use a test simulation procedure to locate potential modification locations. Correction is based on the results of the test simulation procedure together with Boolean function manipulation techniques. Our approach relies on a single source logic design error model, an extension of the one proposed in [1] (Figure 1). An experimental study described in [2] has shown that single design errors as defined in [1] cover 97.8% of all design errors made during a manual resynthesis design procedure. Once a design has been verified to be erroneous ([1], [3]) we apply test–vectors to the functional specification and the gate–level implementation of the circuit that produce erroneous primary output responses. These vectors provide information on the location of possible candidate modifications. A number of observations listed in sections 2 and 3 below, allow us to greatly reduce the set of candidate modification locations in a fast and efficient manner. The correction phase of our algorithm returns a set that contains all possible modifications, if they exist in our logic design error model, that rectify the design. In this paper, the words *correction* and *modification* are used interchangeably.

In [1], Abadir et al presented a single design error model for the DEDC problem (Figure 1). A subset of the model of [1] is adopted by most of the methods proposed for the DEDC problem. This includes [4] and [5]. An error equation at each line is formed and candidate lines that give no solution to this equation are deleted. The work in [8] is also based on Boolean comparisons and applies to multiple errors. Overall, the method does not guarantee to return a solution. In [6] a novel technique for dynamic support for constructing BDDs is presented so that the mem-

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

ory explosion problem is eliminated. No error model is assumed, but their method relies on the existence of the ideal gate–level implementation and heuristics are employed.

Test-vector simulation methods proposed for the DEDC problem include [10], [11], [12], and [13]. In [10], the minterm differences at the output of the circuit under consideration are used to devise applicable hardware at the primary outputs and rectify the design. Gate type changes are then applied and retained if they reduce the size of that hardware, but their method does not guarantee to return a solution. The work of [11] applies to macro–based circuits. Test simulation is used and various counters at the macros, conceptually similar to the counters presented in [13] and the ones presented in this work, are updated and used to screen out candidates that are not suitable for correction. The experimental results of both [10] and [11] are based on small circuits and the run times are not available. Finally, the work of Tomita et al in [12] and [13] uses a restricted version of the error model of [1]. It is based on simulation of IPLDEs (Input Patterns for Locating Design Errors) drawn from the set of values $\{0, 1, X\}$ and generated with the use of BDDs.

We can conclude from the above discussion that most of the methods that consider the DEDC problem are limited in the sense that they use a small subset of the error model of [1] and there is no guarantee that they will return a correct solution even if one exists. In comparison, our algorithm guarantees to return a solution, if such a solution exists in our modification model, and our run time results are better than all the previous methods for locating and correcting single source logic design errors.

The rest of the paper is organized as follows. In the next section we define our single source logic design modification model, together with, formal definitions and the theoretical basis of the algorithm. Sections 3 and 4 explain the error detection and error correction procedures, respectively. Section 5 presents the experimental results and section 6 contains the conclusions.

2 Definitions and Problem Formulation

Once a circuit C has been found erroneous ([1], [3]) we enter the Error Location and Correction stage of our algorithm. The *inputs* to this stage are the *functional specification* F_C of C, an *incorrect gate–level description* G_C of C, and a set of vectors V_{act} drawn from the universe $B = \{0, 1\}$. The set V_{act} is a set of

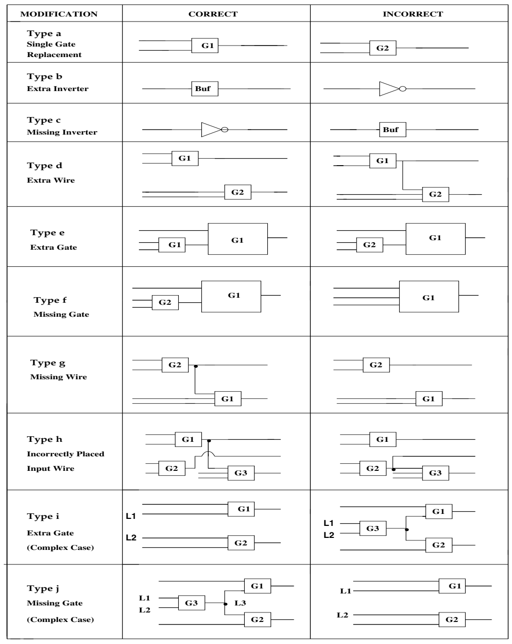

Figure 1: Abadir et al Design Error Model

vectors each of which **activates the inconsistencies** of G_C, that is, $\forall v \in V_{act}$ there is at least one primary output with incorrect (complemented) value when v is simulated. In our experiments, we compiled the set V_{act} by applying test vectors for stuck-at faults([9]) and bridging faults([7]). Abadir et al ([1]) proved that a complete set of test–vectors for stuck-at faults is guaranteed to detect the majority of the design errors and has a very good chance of detecting the remaining ones. An alternative way to compile V_{act} is through Boolean reasoning. An exclusive-or operation can be performed between the function of each erroneous primary output of G_C with its respective correct function from F_C. Every different assignment of primary input values that satisfies the result of this exclusive-or operation is a test that activates the inconsistencies.

The *output* of the algorithm is a list of all the modifications from our design modification model that rectify the circuit. We now proceed to the definition of our **Single Source Modification (SSM)** model:

<u>**Definition**</u> **1** *We define a* **modification** *to be one of the following three types:*

- **Wrong Gate** , *namely types a, b, c, e, f, and i of Figure 1.*

- **Misplaced Wire** , *namely types d, g, and h of Figure 1.*

- **Wrong Gate/Misplaced Wire**, *an occurrence of both previous errors on a gate.*

Definition 2 *An incorrect gate–level description G_C of a circuit C is* **single source correctable** *if there exists a line l in G_C and one modification from Definition 1 on the gate driving l such that, when performed, rectifies G_C.*

Comparing the SSM model to that of [1] we observe that SSM covers all types except type j errors where the error can propagate to the primary outputs *either* through G_1 *or* G_2. In addition, the Wrong Gate/Misplaced Wire case of the SSM model is an extension of the error model presented in [1]. For the SSM model, a complete set of modification locations that could rectify an incorrect gate–level description contains the **actual** modification location and all **equivalent** modification locations since there can be more than one way to synthesize a particular function and rectify the design.

In the discussion that follows, we examine incorrect combinational gate–level descriptions with `NOT`, `BUFFER`, `AND`, `NAND`, `OR` and `NOR` gates. During the execution of our algorithm, we introduce one buffer for every fan-out line of a branch. We say that a line l, fan–in to an `AND` or an `NAND` (`OR` or `NOR`) gate has *controlling value* for input vector v if the value of l is 0 (1). 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, first presented in [13].

Definition 3 *Let vector $v \in V_{act}$ and let $PO_i \in PO$ with* incorrect *output value under v. We define the* **observability measure,** OM_l^i, *of a line l for vector 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 driving two or more branches, then OM_l^i is equal to the minimum value between 1 and $\sum_{l'} OM_{l'}^i$, where l' drives a buffer at the fan-out branch of l.

Definition 4 *We define the* **accumulated observability measure** AOM_l^v *of a line l over a vector $v \in V_{act}$ to be the sum of the observability measures of l for all erroneous primary outputs PO_i for v, that is:*

$$AOM_l^v = \sum_{PO_i} OM_l^i$$

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). Similarly, the accumulated observability AOM_l^v indicates the potential of l to change *all erroneous* primary outputs (for v). The recursive definition of both quantities result in efficient ways for their computation.

Theorem 1 *If l is an actual or equivalent SSM location for an incorrect gate–level description and $v \in V_{act}$ then AOM_l^v is equal to the number of all erroneous primary outputs PO_i for v, that is:*

$$AOM_l^v = \sum_{PO_i} OM_i^i = \#erroneous\ POs$$

The proof of the theorem is by induction on the circuit levels starting from primary outputs towards primary inputs and the use of Definition 3. Due to lack of space the details are omitted. Its significance lies in the fact that it provides a powerful tool to screen out candidate modification locations that have an accumulated observability measure less than the total number of erroneous primary outputs (for v) and, therefore, cannot be an actual or equivalent SSM location.

The initial set of candidate SSM locations, C_{SSM}, is the set of all lines of C. 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 SSM. By applying the above procedure repeatedly on the reduced set C_{SSM} for each vector of V_{act}, we are able to reduce at every step the cardinality of C_{SSM} and guarantee that we do not delete any actual or equivalent candidates. This fact is summarized in the following corollary:

Corollary 1 *If l is an actual or equivalent SSM for an incorrect gate–level description then:*

$$\sum_{v \in V_{act}} AOM_l^v = \sum_{v \in V_{act}} (\sum_{erroneous\ PO_i} OM_i^i)$$

3 Error Detection

This section presents the procedure that implements the theory of the previous section. The **Total Observability Measure (TOM)** and **Inverted Simulation** steps are used to screen out elements of C_{SSM} where no modification that rectifies the design can be applied.

During the TOM process (Figure 2), evaluation of the OM values takes place in a backward fashion, i.e. from primary outputs towards the primary inputs. Every line l has an array of pointers, $l \rightarrow Vgroup$. The size of this array is equal to $\mid V_{act} \mid$ and the i-th entry of this array, $l \rightarrow Vgroup[i]$, will eventually contain the Observability Measure values of l for vector $v_i \in V_{act}$. Initially, all such $Vgroup$ pointers point to NIL. As seen in Figure 2, the first step of the approach is to simulate all vectors of V_{act} and update the respective $Vgroup$ entries at the primary outputs (lines 1–6). This information is then propagated backwards to the internal lines of the circuit (lines 7–8) and the $AOM_l^{v_i}$ value of each line l for every $v_i \in V_{act}$ is computed at lines 9–11. Finally, we screen out the potential modification locations that do not satisfy Corollary 1 (line 12).

Example 1: Figure 3 shows the *incorrect* gate-level implementation of a circuit where G_{12} is intended to be an `NAND` gate. The vectors applied are $V_1 = (1,1,1,0)$ and $V_2 = (1,0,0,0)$ at primary inputs PI_1, PI_2, PI_3, PI_4. The pairs of incorrect/correct values for the input vectors (V_1, V_2) are enclosed in parenthesis and the total observability measure of each line l, $AOM_l^{V_1} + AOM_l^{V_2}$, is encircled when nonzero. The value of *total_aom* is equal to 3 since V_1 propagates the inconsistency at one primary output and V_2 at both primary outputs. Initially, the size of C_{SSM} is 17. After completion of the TOM procedure we have that $C_{SSM} = \{PI_3, G_{12}\}$ since all other lines have a total observability measure less than *total_aom*.

During the simulation step of the TOM process, we create a linked–list of bit-vectors at each line l of C. The i-th bit of this list for a line l contains the value of l when $v_i \in V_{act}$ is simulated. For Inverted Simulation, we complement the values of the faulty bit-list for every remaining candidate $l \in C_{SSM}$ separately and we resimulate 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_{SSM}.

Example 2: The numbers in the three boxes pointing to lines in Figure 3 are the new values of the lines at the fan–out cone of G_{12}, when Inverted Simulation is performed for this candidate. These lines are the *only* three lines visited during Inverted Simulation for G_{12}. Since both primary output responses are correct for complemented values of the bit-list, G_{12} remains in C_{SSM}. This is not true for PI_3, where both input values produce erroneous primary outputs as shown in Example 1. This is reported by Inverted Simulation and PI_3 is removed from the candidate list.

4 Error Correction

In this part of our algorithm we use an *error equation* to obtain the applicable corrections. The results of the test simulation procedure of V_{act} are used to prune the search space and speed up the computation.

The following lemma, adapted from [4] and [5], provides a necessary and sufficient condition for an error candidate to be an actual or equivalent SSM location. It also gives information on potential correction procedures.

Theorem 2 *Let line l be a candidate modification location. Let X be the set of primary inputs of F_C and G_C. Let $f_i(X)$ be the global function of the i-th primary output for F_C and $g_i(X,z)$ the global function of the same primary output for G_C when the function at l has been replaced by a new primary input variable z. Let the Error Equation at l be defined as: $EQ^l(X,z) = \sum_{i \in ErroneousPOs} f_i(X) \oplus g_i(X,z)$. Then the following two conditions hold:*

- *If $EQ^l(X,0)EQ^l(X,1) = 0$ then there exists some correction on line l that rectifies the design (Condition 1).*
- *If $EQ^l(X,0)EQ^l(X,1) = 0$ then any new function z at l that corrects G_C belongs to the interval $[EQ^l(X,0), \overline{EQ^l(X,1)}]$ (Condition 2).*

Once a candidate passes the screening test of Condition 1, we check if a *Wrong Gate* correction (Definition 1) can be applied and satisfy Condition 2. If there is one, the algorithm reports the correction and terminates, otherwise we look for a *Misplaced Wire* correction. If a modification of type d rectifies the design then removing the extra wire will make l satisfy Condition 2. For the remaining two types of errors, we need to check every line l' of the circuit such that there is no path from l to l'. In order to reduce the search space and perform this step efficiently, we use the faulty bit-list we compiled during the simulation of V_{act}. Let G be the gate that drives l. If l', as a new fan-in to G or replacing an existing fan-in of G, cannot produce complemented faulty bit-vector values at l, then it is disregarded since Condition 2 is violated. If it does create complemented values, we explicitly check Condition 2 and terminate the program if a solution is found.

If both Wrong Gate and Misplaced Wire correction types fail to provide a solution, we try a combination of both (*Wrong Gate/Misplaced Wire* case). In detail, we first compile a list of all Wrong Gate corrections from Definition 1 and then apply the technique of Misplaced Wire on each member of the list. If no correction is found during this step we conclude that the circuit cannot be rectified with one correction from the SSM model.

It is also straightforward to see that if the algorithm is not terminated after a correction is found then a list of *all* the corrections that rectify the design can be obtained one by one.

5 Experimental Results

We ran the above algorithm on a Sparc 20 workstation and tested it on ISCAS'85 benchmark circuits for the three types of design errors of the SSM model, the Wrong Gate (WG), Misplaced Wire (MW) and Wrong Gate/Misplaced Wire(WG/MW) ones. The locations where the modifications were injected were selected randomly. We repeated the experiment 20 times for *each* of the above errors and required the algorithm to return *all* possible corrections from our modification model. The average values of the results are reported in Table 1. All run-times are in seconds.

As mentioned earlier, the set of initial candidates C_{SSM} is equal to the set of all lines of the circuit (Table 1, Column 2). The first step of our algorithm is to simulate a set of test-vectors for bridging faults and stuck-at faults originating from [7] and [9] and record the vectors that activate the inconsistencies. The average size of V_{act} needed for an efficient solution to DEDC is given in column 5. Columns 3 and 4 contain the hit-ratios of vectors activating design errors versus the total number of input test vectors we simulated. The high hit-ratio of those vectors support the argument in [1] that a complete stuck-at fault test set provides satisfactory verification for design errors as well. Column 6 contains the average time needed to locate candidate error locations (TOM + Inverted Simulation). Columns 7, 8, and 9 contain the average run times of Wrong Gate, Misplaced Wire and Wrong Gate/Misplaced Wire corrections, respectively. The values of column 7 include the time needed to create the error equation at the error locations. Column 10 of Table 1 shows the maximum run-time reported during our experiments.

As a next step, we applied the above algorithm to incorrect gate–level implementations that have been corrupted by two design errors. Since there can be different ways to synthesize a particular function, it is possible that one SSM is sufficient to correct a design that has been corrupted by two design errors. If such the case, our algorithm guarantees to return a solution. In our experiments we injected 20 pairs of errors where the first error of each pair was in the dominator set ([4]) of the second. The last two columns of Table 1 contain the hit-ratio of our algorithm and the average run time to return with success or failure, respectively. The algorithm returns with a failure when no SSM modification location exists to correct a design corrupted by two design errors.

6 Conclusions

In this paper, we first defined a design error scenario for combinational circuits where certain modifications on a line of the circuit are sufficient to rectify the corrupted gate–level design. An efficient synthesis algorithm that returns all possible actual and equivalent single modification locations with their respective corrections was also presented. The algorithm is based on test simulation and Boolean function manipulation techniques. The experimental results show the robustness of the proposed algorithm which guarantees to return a solution if such a solution exists in our modification model.

Acknowledgments. The authors would like to thank Pi-Yu Emerald Chung and Nikos Bellas for their helpful comments.

References

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

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

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

[4] 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.

[5] P.-Y.Chung, and I.Hajj. "Diagnosis and Correction of Multiple Logic Design Errors in Digital Circuits". *IEEE Trans. on VLSI Systems,* accepted.

[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.Lee, I.Hajj, E.Rudnick, and J.Patel. "Genetic–Algorithm–Based Test Generation for Current Testing of Bridging Faults in CMOS VLSI Circuits". *Proc. of IEEE VLSI Test Symposium, pp.456–462,* 1996.

[8] C.-C. Lin, K.-C. Chen, S.-C. Chang, and M.M.Sadowska. "Logic Synthesis for Engineering Change". *Proc. of ACM/IEEE DAC, pp.647–652,* 1995.

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

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

[11] I.Pomeranz and S.M.Reddy. "On error correction in macro-based circuits". *Proc. of IEEE/ACM ICCAD, pp.568–575,* 1994.

[12] M.Tomita,H.-H.Jiang,T.Yamamoto, and Y.Hayashi. "An algorithm for locating design errors". *Proc. of IEEE/ACM ICCAD, pp.468–471,* 1990.

[13] 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.