High-Level Design Error Diagnosis Using Backtrace on Decision Diagrams

Jaan Raik, Urmas Repinski, Raimund Ubar, Maksim Jenihhin, Anton Chepurov

Tallinn University of Technology, Estonia

Abstract. *The paper proposes a method for locating design errors at the source-level of hardware description language code using the design representation of High-Level Decision Diagram (HLDD) models. The method is based on backtracing the mismatched and matched outputs of the system under verification on HLDDs. Experiments on a set of sequential register-transfer level benchmarks show that the method is capable of locating the design errors injected with a high accuracy and a short run time. In fact all the errors injected in the experiments were identified as top suspects by the proposed diagnosis algorithm.*

1 Introduction

Designing a microelectronic chip is a very expensive task and excessive design costs are the greatest threat to continuation of the semiconductor industry's growth [1]. It is a well acknowledged fact that verification is forming a major part in the total product design cycle [2] and this trend continues. At the same time when there have been numerous research works on verification methods identifying the occurrences of errors, the problem of diagnosing the causes of errors and correcting them has been largely neglected. Yet a large part of the verification cycle is consumed inside the design loops between debugging and correction. It is estimated that fault location and correction constitute roughly half of the total time spent on verification and debug [2]. Verification and debug (i.e. assuring the correctness of the design), in turn, represent the main reason of the excessive costs accounting for about 70 % of design expenses [2]. Location and correction costs therefore form about 1/3 of the total design time [3].

Design error diagnosis for combinational circuits has been thoroughly studied for two decades. There exist, both, fault-model-based [4, 5] and fault-model-free [6] approaches. There have been attempts to generalize the above methods for design error diagnosis of sequential circuits [6, 7], resulting in serious scalability problems.

Previous works on error diagnosis for high-level models like the Register-Transfer Level (RTL) are based on the work by Smith et al. [8]. In the error diagnosis technique [8], three components are added to the netlist: (1) multiplexers, (2) test vector constraints, and (3) cardinality constraints. The whole circuit is then converted to conjunctive normal form, and a satisfiability solver (SAT) is used to perform error diagnosis. These components are added temporarily for error diagnosis only and will not appear in the netlist produced. There is a range of works extending this idea of SAT-based diagnosis including [9, 10].

However, these methods reduce the diagnosis problem to SAT, which is an NP-complete problem. Although SAT engines are being constantly developed and improved, there is a clear limit to the circuit size where this approach is no more applicable. Current paper considers a different approach relying on design error diagnosis utilizing HLDD simulation that executes in polynomial time. This means that much larger designs could be potentially handled by the proposed method.

The SAT-based approaches [8-10] perform the diagnosis on logic-level formal engines, i.e. SAT solvers. Current paper utilizes HLDD simulator as a source-level reasoning engine for the diagnosis process. In our case the engine operates directly on the register-transfer level. This results in a readable diagnostic feedback and is therefore better understandable to the engineer than logic-level debug information provided by previous methods.

In this paper we propose a new method for locating design errors at the source-level of hardware description language code using backtrace of mismatched and matched outputs of the system under verification on High-Level Decision Diagram (HLDD) models. The paper is organized as follows. Sections 2-3 provide the preliminaries of HLDD modelling and simulation. Section 2 defines the HLDD data structure. Section 3 presents simulation on HLDDs. In Section 4, we introduce the new method for design-error diagnosis based on HLDD backtrace. Section 5 gives the experimental results and Section 6 concludes the paper.

978-1-4244-8973-2/10/$26.00 ©2010 IEEE

2 High-level decision diagrams

In this Section we define the HLDD data structure. Consider a digital system (Z, F) as a network of subsystems or components, where Z is the set of variables (Boolean, Boolean vectors or integers), which represent connections between components, primary inputs and primary outputs of the network. Let $Z = X \cup Y$, where X is the set of function arguments and Y is the set of function values where $Q = X \cap Y$ is the set of state variables. $D(z)$ denotes the finite set of all possible values for $z \in Z$ and $D(Z')$ is the set of all possible vectors in some variable set $Z' \subseteq Z$. Obviously, if $Z' = \{z_1, \ldots, z_n\}$ then $D(Z') = D(z_1) \times \ldots \times D(z_n)$. Let F be the set of discrete functions: $y_k = f_k(X_k)$, where $y_k \in Y$, $f_k \in F$, and $X_k \subseteq X$ (k iterates over all elements in F).

Definition 1. High-level decision diagram representing a function $f_k : D(X_k) \rightarrow D(y_k)$ is a directed acyclic multigraph $G = (V, E)$ with a single root node and a set of terminal nodes where:

- Each non-terminal node is labeled by some input or control variable $x \in X$. We shall denote the variable of node v by x_v.
- Each terminal node w is labeled by some function $g_w : D(X_w) \rightarrow D(y_k)$, where $X_w \subseteq X_k$.
- Each edge $e = (v, u)$ is labeled by some constant $c_e \in D(x_v)$.
- Each two edges $e_1 = (v, u_1)$ and $e_2 = (v, u_2)$ starting from the same source node are labeled by different constants $c_{e_1} \neq c_{e_2}$.
- If the node v is labeled by x_v then the number of edges starting from this node is $|D(x_v)|$.

Remark 1. Each BDD is HLDD as well, with two terminal vertices labeled by constant functions 0 and 1, and $D(x) = \{0, 1\}$ for every variable x.

3 Simulation on HLDD models

HLDD models can be used for representing digital systems. In such models, the non-terminal nodes correspond to conditions or to control signals, and the terminal nodes represent data operations, variables or constants. When representing systems by decision diagram models, in general case, a network of HLDDs rather than a single HLDD is required. During the simulation in HLDD systems, the values of some variables labeling the nodes of an HLDD are calculated by other HLDDs of the system.

Simulation on high-level decision diagrams takes place as follows. Consider a situation, where all the node variables are fixed to some value. According to these values, for each non-terminal node a certain output edge will be chosen which enters into its corresponding successor node. As mentioned above, such connections between nodes are referred to as the activated edges under the given values. Succeeding each other, activated edges form in turn activated paths. For each combination of values of all the node variables there exists always a corresponding activated path from the root node to some terminal node. Let us call this path the main activated path. The simulated value of the variable represented by the HLDD will be the value of the variable labeling the terminal node of the main activated path.

Algorithm 1 presents simulation on HLDD models. The simulation process starts in the root node v_0 (line 2 of the algorithm). The node $v_{Current}$ is iteratively replaced by its successor nodes selected according to the value of $x_{v_{Current}}$ (line 4). In order to represent feedback loops in the system, in the RTL style, the algorithm takes the previous time-step value of variable x_k labeling a node v_i if x_k represents a clocked variable in the corresponding HDL (lines 5, 6). In the case of behavioral HDL coding style HLDDs are generated and ranked in a specific order to ensure causality. For variables x_k labeling HLDD nodes the previous time step value is used if the HLDD calculating x_k is ranked after current decision diagram (lines 5, 6). Otherwise, the present time step value will be used (line 8). Simulation ends when a terminal node is reached and the variable y corresponding to the simulated HLDD G_y is assigned the value $x_{v_{Current}}$ (line 12).

Algorithm 1. HLDD simulation

```
1:   SimulateHLDD($G_y$)
2:       $v_{Current} = v_0$
3:       While $v_{Current} \notin V^T$
4:           $x_k = x_{v_{Current}}$
5:           If $x_k$ is clocked or
              its HLDD is ranked after G then
6:               Value = previous time-step
                             value of $x_k$
7:           Else
8:               Value = present time-step
                             value of $x_k$
9:           End if
10:          $v_{Current} = v_{Current}^{Value}$
11:      End while
12:      Assign $y = x_{v_{Current}}$
13:  End SimulateHLDD
```

4 HLDD backtrace for error location

In this subsection we first present the algorithm for diagnostic tree generation using backtrace on HLDD models. Then, two analysis steps are introduced to perform error location on the set of diagnostic trees generated.

Algorithm 2 presents the recursive diagnostic tree generation on HLDDs. The process starts from the primary outputs (Line 2) and from each clock-cycle (Line 3). Subsequently, the diagnostic tree is recursively generated using the function *RecursiveTreeGeneration*.

Algorithm 2. HLDD-based diagnostic tree generation

```
1:  GenerateDiagnosticTree()
2:      For each primary output $G_O$ in the model
3:          For each time-step t
4:              $\delta(G_O, t) = \varnothing$
5:              RecursiveTreeGeneration($G_O$, t, $\delta$)
6:          End for
7:      End for
    End GenerateDiagnosticTree

    RecursiveTreeGeneration($G_y$, t, $\delta$)
8:      SimulateHLDD(Gy)   // See algorithm 1!
9:      For each $v_i$ at the main activated path
10:         If variable $x_k = x_{v_i}$ at-time step t is not in $\delta$ then
11:             Add $x_k$ to $\delta$
12:             If $x_k$ is not a primary input then
13:         RecursiveTreeGeneration($Gx_k$, t, $\delta$)
14:             End if
15:         End if
16:     End for
     End RecursiveTreeGeneration
```

Algorithm 2 generates a separate diagnostic tree $\delta(G_O, t)$ for each output diagram G_O at each clock-cycle t. The resulting diagnostic tree δ is a set of pairs (x_i, t_j) that show at which time-steps t_j the variable x_i was backtraced.

In the following, two analysis steps that could be implemented for locating the design error are presented. In order to perform the analysis, let us partition the set of all diagnostic trees $\Delta = \delta_k(G_O, t)$ into failing diagnostic trees Δ_F and passing diagnostic trees Δ_P. A diagnostic tree is failing if $\delta_k(G_O, t)$ if the simulated value of output variable $o \in Y$ differs on the faulty design at time-step t differs from the corresponding value of the golden device. Otherwise, δ_k is called a passing sequence.

Diagnosis step 1:

For each variable x_i count the number c_{FAILED} of failing diagnostic-trees $\delta_k \in \Delta_F$, where x_i is present at least in one of the pairs (x, t) of δ_k. Select the variables x_{step1} receiving the highest score c_{FAILED} as the list of suspected faults $X_{suspected}$.

Diagnosis step 2:

Perform step 1. For each variable $x_{step1} \in X_{suspected}$ count the number of passing diagnostic-trees $\delta_l \in \Delta_P$ c_{PASSED} , where x_{step1} is present at least in one of the pairs (x, t) of δ_l. Compute the score $c_{TOTAL}=c_{FAILED}$ / c_{PASSED} for variables x_{step2}. Select the variables x_{step2} receiving the highest score c_{TOTAL} as the new list of suspected faults.

Step 1 is more exact as it can be easily proven that at least one of the variables x_v that is labeling a vertex v along one of the main activated paths in simulated HLDDs must be also the cause of the error. However, step 2 may be unavoidable in order to guarantee a good diagnostic resolution, especially if the number of failing sequences is one or very small. In fact, the experiments presented in Section 5 fully confirm this observation.

5. Experimental results

Table 1 presents the main characteristics of the benchmarks used in the experiments and their respective test sets. The benchmarks include the Greatest Common Divisor (*gcd*) and the Differential Equation (*diffeq*) examples from the HLSynth92], HLSynth95 academic benchmarks suite, respectively. *risc* is a processor example from a FUTEG research project. In addition, a commercial design core for circular redundancy check (*crc*) was selected [11]. The test stimuli for the academic benchmarks were generated by a hierarchical test pattern generator Decider [12] while for *crc* the provided functional test bench was applied. The second column reports the system complexity in terms of the number of HLDD nodes. The third column represents the number of functions in the design. The fourth column shows the number of stimuli in the test suite. Finally, column 5 reports an average run time per inserted fault.

Table 1. Benchmarks and their test sets

design	# nodes	# fun.	# test stimuli	run time, s
gcd	25	4	4000	20
diffeq	39	9	16855	800
risc	61	16	4000	40
crc	232	27	42	0.8

In Table 2, the design error location experiments are provided. We injected faults into the design by randomly mutating a function one-by-one, so that during each diagnosis run only one function was mutated. The column ‘success rate’ shows the ratio of the times the actual location of the mutation achieved **the highest rank** in relation to all diagnosis runs. The column ‘average resolution, # suspects’ reports the average number of suspects that received the highest score. Here, the diagnostic resolution is very good for step 2 and two or more times worse for step 1. The same trend applies to the worst resolution, which reports the worst case suspected fault list size over all the faults injected. The final column reports the run times achieved on a PC, Dual-Core CPU, 2.6GHz, 3.25GB RAM, Windows XP operating system are provided. This time includes both performing step 1 and step 2 of the diagnosis algorithm. As it can be seen, the run times are very different. They do not only depend on the circuit size but also the number of vectors and the sequential depth of the designs. The run time for step 1 is actually very much shorter than the time for steps 1 and 2 combined, because in step 1, only mismatched outputs have to be backtraced. Note, that the speed of the algorithm could be considerably increased in the future because current implementation backtraces full trees for each output at each clock-cycle, while it is possible to avoid multiple backtrace of already analyzed sub-trees in the diagnostic tree.

Table 2. Design error location experiments

design	success rate, ratio correct locations		average resolution, # suspects		worst resolution, # suspects	
	step1	step2	step1	step2	step1	step2
gcd	**4/4**	**4/4**	2.25	**1**	3	1
diffeq	**9/9**	**9/9**	3.31	**1.88**	6	3
risc	**16/16**	13/16	7.97	**1.44**	12	3
crc	**27/27**	23/27	15.09	**4.56**	24	13

It is important to note that step 1 always gives the injected error the top score in the suspected faults list. Therefore the method is accurate even though designs with sequential loops were targeted. Though mutation-based fault insertion was applied in the experiments, the method is open with respect to the fault model. It is also not limited to the single fault assumption. In the future, we plan to apply the HLDD backtrace based diagnosis method to multiple faults.

6. Conclusions

In this paper, a diagnosis method for locating design errors at the source-level of hardware description language code using the design representation of high-level decision diagrams is proposed. The method is based on backtracing the mismatched and matched outputs of the system under verification on HLDDs. Experiments on a set of sequential register-transfer level benchmarks show that the method is capable of locating the design errors injected with a high accuracy and a short run time. Current paper relies on HLDD simulation that executes in polynomial time. Therefore, larger designs could be potentially handled by the proposed method than with the existing formal approaches.

References

[1] *International Technology Roadmap for Semiconductors*, Design, 2007 edition http://www.itrs.net/Links/2007ITRS/Home2007.htm

[2] William K. Lam, *Hardware Design Verification: Simulation and Formal Method-Based Approaches*, Pearson Education Inc., 2005, 585 p.

[3] *PROSYD (Property-Based System Design) project deliverable*, FP6 funded STREP, 2004, http://www.prosyd.org.

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

[5] M.S. Abadir, J. Ferguson,T.E. Kirkland, "Logic design verification via test generation," *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 1, Jan. 1988.

[6] Ali, M. F., Safarpour, S., Veneris, A., Abadir, M. S., and Drechsler, R. Post-verification debugging of hierarchical designs. *In Proceedings of ICCAD*, pp. 871-876. 2005.

[7] A. Wahba, D. Borrione. Design error diagnosis in sequential circuits. *Lecture Notes In Computer Science*; Vol. 987, pp. 171 – 188, Springer, 1995.

[8] A. Smith, A. Veneris and A. Viglas, “Design Diagnosis Using Boolean Satisfiability”, *Proc. Asia and South Pacific Design Automation Conference (ASPDAC)*, 2004, pp. 218-223.

[9] Görschwin Fey, Stefan Staber, Roderick Bloem, Rolf Drechsler. Automatic Fault Localization for Property Checking. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(6): 1138-1149, 2008.

[10] Kai-hui Chang, et al., "Automatic Error Diagnosis and Correction for RTL Designs", *Proc. High-Level Design and Validation Workshop (HLDVT)*, Irvine, CA, November 2007.

[11] EU FP6 IST STREP VERTIGO project benchmarks. URL:http://www.vertigo-project.eu/

[12] Jaan Raik, Raimund Ubar. Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations. *JETTA*, Kluwer Academic Publishers. Vol. 16, No. 3, pp. 213-226, June, 2000.