RTL Test Generation via Fault Insertion and Hybrid Satisfiability Solving

Weimin Wu

Department of Computer Engineering, Beijing Jiaotong University

Abstract

Test generation at RTL (Register-Transfer Level) is a challenging task because bit and word variables co-existent and the high-level functional units impose more complex constraints. We propose an effective way to the problem. In our method, given the circuit as well as the fault point to be checked, we first construct a new circuit, miter, by fault insertion as well as miter reduction techniques. Then we solve the constraints of the miter by EHSAT, an efficient hybrid satisfiability solver to obtain the required test vectors. Experimental results demonstrate the effectiveness of our method.

Keywords: RTL, fault, test generation, satisfiablity.

1. Introduction

Test vectors are critical in functional validation. Contrary to randomly generated ones, the test vectors generated by ATPG (automatic test pattern generation) tools apparently have higher efficiency in size and coverage. However, the ability of current ATPG tools is being overwhelmed by larger and more complex designs. One of the viable approaches to this problem is to do test generation at higher levels of abstraction [1].

By high-level we usually mean behavioral level and RTL (register transfer level), where the later receives more attention because RTL designs contain useful high-level information while still maintain some circuit structure.

So far, the research efforts in dealing with the complexity in RTL functional test generation focus on either test strategy or problem solving engines used. Test strategy includes techniques such as test coverage [2], test model such as ADD (assignment decision diagram) [3], and hierarchical test generation [4]. Ref [3] presented an algorithm to automatically generate test vectors from RTL circuits that target stuck-at faults in gate-level circuits, which operates on a data structure called assignment decision diagram. In [4], three levels of modeling are proposed, that is, high-level DD (decision diagrams), low-level Boolean differential equations, and medium-level DD.

Until now the researchers have used various engines for test generation, such as LP (linear Programming) [5], ATPG [6], SAT [7], etc. The RTL circuits can be modeled in a unified way such as LP, but efficiency is the main drawback. If we want to apply Boolean ATPG or SAT, the RTL circuits have to be synthesized into gate-level circuits, which may result in tremendous circuit sizes. To solve the problem, [7] proposed to solve the Boolean constraints in 3-SAT and arithmetic constraints in LP, while in [8] the corresponding solvers used are word-level ATPG and modular arithmetic solver. Furthermore, there are still some works on extending traditional Boolean solver to deal with mixed bit/word constraints. Ref. [9] extended the DPLL procedure of Boolean SAT to cope with hybrid bit/word constraints. Ref. [10] follows the similar idea but implements the procedure more

completely and thus improves the performance fairly.

In this paper, we will propose a RTL test generation method that is based on solving the satisfiability problem of the modified RTL circuit by EHSAT, a RTL constraint solver [10] we have developed. The miter, which is a new circuit created from the original circuit and by fault insertion, will compile the difference between faulty and fault-free circuits, and if solved, will generate the test vector that can identify the difference.

The rest of the paper is organized as: In Section 2, we will give an overview of our algorithm. Then we discuss how to create the miter by fault insertion in Section 3. Following we will introduce the hybrid satisfiability solver EHSAT in Section 4. In Section 5, experiments are conducted and results analyzed. Section 6 concludes the paper.

2. Overview of the method

The principle of satisfiability-based RTL test generation is illustrated in Fig.1. In the figure, the MUX is a multiplexor, = (equal) and > (great than) are comparators, + is an arithmetic unit (adder), AND and OR are Boolean gates, W1, W2, W3, W4 are word inputs, B1 and B2 are Boolean inputs, OUT is an output.

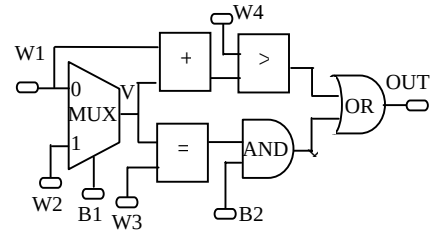


Fig. 1: A simple RTL circuit

Suppose we want to find a test vector that can detect a possible fault in the output of AND (crossed point). That is, if the test vector is used as input to the circuit, the OUT of the correct and faulty circuit will output different values.

One approach to the problem is to duplicate the original (faulty) circuit, and by adding some monitor circuits, construct a new circuit, called miter, as shown in Fig.2. The signal of the fault point in original circuit is inverted and replaces the corresponding signal in the duplicated circuit. The outputs of original and duplicated circuits become inputs of a XOR gate, with MOUT as the global output. For efficiency reason, the original circuit need not be completely duplicated. In fact, we need only duplicate those units as well as connected lines that would be affected by the fault. For the circuit in Fig. 1 and the given fault point, duplicating only OR is enough, while other units and lines can be shared between original and duplicated circuits.

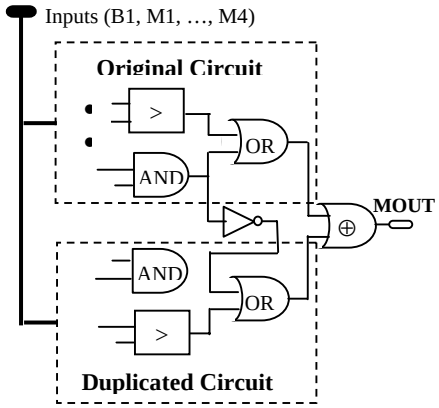


Fig. 2: The miter for test generation

For the miter in Fig.2, we conclude that the there is a test vector that makes faulty and fault-free circuit output different values iff we can find an assignment of inputs that makes MOUT evaluate 1.

The miter implies a set of constraints, which if solved, will result in the test vector. The constraints come from the input/output relation of each unit as well as other requirements on specific variables, such as MOUT=1. These constraints are hard to solve because hybrid bit/word datatypes as well as more complex input/output relations must be dealt with.

For above idea to be effectively implemented, we must construct the miter with minimum size and solve the constraints of the miter efficiently. The test generation algorithm follows two steps:

(1) According to specific fault point to be checked, a miter with minimum possible size is constructed. The miter can identify the difference between faulty and fault-free circuit via outputting 1 at MOUT, which is realized by fault insertion. The size of the miter is reduced by topology analysis.

(2) The constraints implied by the miter (determined by the units and interconnections it contains), with MOUT=1 as the initial condition, are solved by a RTL hybrid satisfiability solver, EHSAT. If satisfiable, the resulting input is the test vector. Otherwise, the fault is unobservable at the output.

3. Miter construction

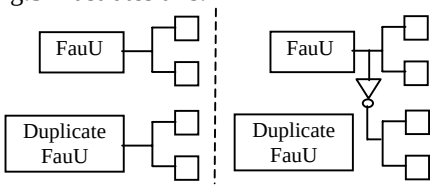
First of all, a duplication of the original circuit is created, which shares primary inputs with the original circuit. The outputs of the two circuits are XORed by a XOR gate, whose output is MOUT. Following, the two major tasks are fault insertion and meter reduction.

3.1. Fault insertion

The units (or components) of RTL circuits can be classified into three kinds: Boolean gates (such as AND, OR, NOT), arithmetic units (such as ADD, SUB), and interface units (such as MUX, CMP).

For any unit whose output variable (or line) needs to be tested for a fault, we call it FauU, fault insertion is operated just on FauU's output. According to the datatype of FauU's output variable, we will use different fault insertion methods.

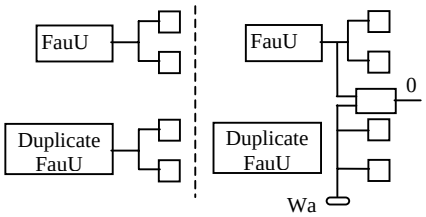
If the FauU is a Boolean gate or a compare gate, that is, it has Boolean output, an inverter will be inserted at FauF's output, and all units in the duplicated circuit that are driven by the FauU are now modified to be driven by the inverter. Fig.3 illustrates this.



(a) Before fault insertion (b) After fault insertion

Fig. 3: Fault insertion for Boolean Signals

If the FauU is an arithmetic unit or a MUX which has a word-level output, an equal comparator (=) as well as a word variable is added. The two inputs of the comparator are FauU's output and the newly added word variable (Wa), and the output of the comparator is set with zero. Also, all units in the duplicated circuit that are driven by the FauU are now modified to be driven by Wa. Fig.4 illustrates this.



(a) Before fault insertion (b) After fault insertion

Fig. 4: Fault insertion for word signals

After fault insertion, it is guaranteed that the generated test vector will differentiate the faulty and fault-free circuit.

3.2. Miter reduction

The reason that we can reduce the size of the miter is that some units are not affected by the FauU, thus can be shared between original and duplicated circuits.

Only the descendants of the FauU needed to be duplicated in building the miter. A unit is a descendant of the FauU if there is a set of successive driving lines

that connect the FauU to the unit. The procedure of computing all descendants for a FauU is as:

(1) Create a set *Des* with only one element, FauU;

(2) If *Des* is empty, stop; Otherwise, take a element, say, *elem*, from *Des*;

(3) Mark *elem* with *visited*;

(4) For each unit driven by *elem* and has not been marked *visited*, put the unit in *Des*;

When the procedure completes, all units that are marked *visited* are descendants of FauU.

For the example in Fig.1 and Fig.2, only OR is a descendant of AND which needs duplicating, other units and lines can be shared. The reduced miter is shown in Fig.5.

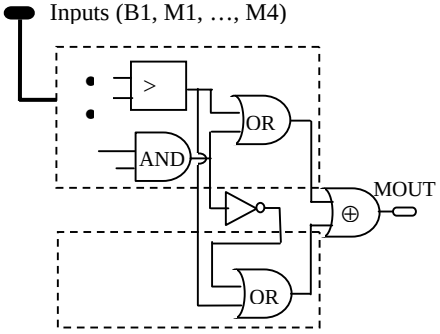


Fig. 5: The reduced miter

4. RTL satisfiability solving

This section is an overview of EHSAT, an efficient RTL satisfiability solver we have developed [10], which has been used as a stand-alone tool in our work. EHSAT adopts a complete extended DPLL procedure with conflict-driven learning, as well as several other techniques for improving performance.

4.1. Extended DPLL procedure

The schematic of extended DPLL procedure is shown in Fig.6.

```
Procedure EDPLL ()
  while (decide ( ) = = SUCCESS)
    while (hcp ( ) = = CONFLICT)
      int (blevel = analyze_conflicts ( );
      If (blevel<0) return UNSAT;
    else backtrack (blevel);
  return SAT;

Procedure analyze_conflicts ( )
  If ( dlevel ( ) = = 0) return -1;
  return First  Bit  UIP ( );
```

Fig. 6: The extended DPLL Procedure

To facilitate searching, both interval and bit-vector representations are kept for word variables. For example, a 3-bit word variable will have initial interval [0, 7] and bit-vector xxx. To avoid splitting the interval, decisions on a word variable is only made on the most significant bit (MSB). So if we assign the MSB of the 3-bit word to 0, its interval becomes [0, 3]; if assign to 1, the interval will be [4, 7].

The *decide()* procedure also uses Variable State Independent Decaying Sum (VSIDS) heuristic rule [11]. Every variable has an activity number, which will shrink to half periodically. When conflict occurs, all variables that are involved in conflict will have this number increased. A variable with larger activity number has higher priority in decision assignment.

EHSAT also uses hybrid constraint propagation, which contains two parts. One is Hybrid-Two-Literal-Watching, which is an extension of zChaff's idea [11], whose main benefit is lessening the workload during backtracking. The other part is predicate-based interval reasoning. One example of interval reasoning is: if X=[2, 4], Y=[3, 5] and X>Y, we immediately conclude X=[4], Y= [3].

4.2. Conflict-driven learning

In EHSAT, conflict-based learning is implemented using an implication graph and First Bit Unique Implementation

Point (IBUIP) scheme, which are adapted from their Boolean versions in [11][12].

When a conflict occurs, the implication graph is used to find the assignments that are directly responsible for the conflict. Through a negation operation, these assignments form a learned clause, which will be added into the clause database. IBUIP helps implement the non-chronological backtracking. The added learned clause prunes the search space so that the solver won't repeat the same conflicts in future searching.

5. Experiments

We implemented our algorithm in C and tested it with some examples provided along with the RTL SAT tool hdpll [9]. The characteristics of the circuits are shown in Table 1.

Table 1. The characteristics of the test cases

Circuits	characteristic		
	#Boolean	#word	#PI / #PO
B01_3	33	16	3 / 1
B01_5	97	46	7 / 1
B01_10	257	121	17 / 1
B01_15	417	196	27 / 1
B01_20	577	271	37 / 1
B02_5	91	86	5 / 1
B02_10	241	226	10 / 1
B02_15	391	366	15 / 1
B02_20	541	506	20 / 1

Boolean / #word: number of Boolean/word units.
#PI / #PO: number of primary inputs/outputs.

Because efficiency is the major concern, we will only investigate the performance issues in the experiments. For an instance, all possible fault point will be considered for test generation, and the global performance is recorded.

We conduct the experiment as: for each instance, we test every possible fault-point for it in topological order, and summarize the results to get the global performance. Table 2 gives the results.

As expected, as circuit size increases, the average time used for a test generation also increases. However, performance degradation does not match circuit scaling. We can use TPU (time-per-unit) as an intuitive measurement, defined as:

$$TPU = \frac{Nu}{Ta} = \frac{Nb + Nw}{Ta} \quad (1)$$

where *Nu*, *Nb*, *Nw* are the numbers of units, Boolean units, and word-level units respectively. *Ta* is the average run time. The TPU results are shown in Table 3.

Table 2. Performance of the TG method

Circuits	Performance		
	#TG	Total time	Average time
B01_3	48	6.07	0.13
B01_5	142	18.14	0.13
B01_10	377	86.81	0.23
B01_15	612	200.27	0.33
B01_20	847	353.06	0.42
B02_5	176	21.16	0.12
B02_10	466	103.19	0.22
B02_15	756	259.22	0.34
B02_20	1046	476.17	0.46

TG: the number of test generations performed.
Total and Average times are in seconds.

Table 3. TPU measurement of the algorithm

Circuits	Measurement		
	Nu	Ta (s)	TPU
B01_3	49	0.13	0.00265
B01_5	143	0.13	0.00091
B01_10	378	0.23	0.00061
B01_15	613	0.33	0.00054
B01_20	848	0.42	0.00050
B02_5	177	0.12	0.00068
B02_10	467	0.22	0.00047
B02_15	757	0.34	0.00045
B02_20	1047	0.46	0.00044

We can see that for both B01 and B02, as circuit size increases, TPU will decrease. This implies that the run time increases slower than circuit size.

Another observation that demonstrates the efficiency of our method is that even for fairly large circuits, such and B01_20 and B02_20, one time test generation consumes only about a half second.

In summary, our algorithm shows good robustness and efficiency.

6. Conclusion

The method proposed in this paper is rather competitive both in generality and in efficiency. For generality, the fault in-

sertion and miter reduction techniques are general-purpose and applicable to any RTL designs expressed as interconnected units. Meanwhile, the efficiency is largely owed to EHSAT.

Our future effort will consider the sequential behavior of RTL designs and extend present work to sequential test generation. Coverage issues will also be included as the measurement of test quality.

7. Acknowledgments

The research in the paper is supported by National Science foundation of China (grant number 60673034), and is also partly supported by Science and Technology foundation of Beijing Jiaotong University (grant numbers 2007XM011 and 2008RC002). These supports are gratefully acknowledged. However, Shujun Deng, the developer of EHSAT, deserves special appreciation.

8. References

[1] G. Jervan, Z. Peng, O. Goloubeva, M. Sonza Reorda, M. Violante, "High-Level and Hierarchical Test Sequence Genaration", *IEEE International Workshop on High Level Design Validation and Test*, 2002, pp.169-174.

[2] F. Fallah, P. Ashar, S. Devadas, "Functional Vector Generation for Sequential HDL Models Under an Observability-Based Code Coverage Metric", *IEEE Trans. VLSI*, vol.10, no.6, pp.919-923, 2002.

[3] I. Ghosh, M. Fujita, "Automatic Test Pattern Generation for Functional Register-Transfer Level Circuits Using Assignment Decision Diagrams", *IEEE Trans. Computer-Aided Design*, vol.20, no.3, pp.402-415, 2001.

[4] R. Ubar, J. Raik, "Efficient Hierarchical Approach to Test Generation for Digital Systems", *International Symposium on Quality Electronic Design*, 2000, pp.189-195.

[5] Z. Zeng, P. Kalla, M. Ciesielski, "LPSAT: A Unified Approach to RTL Satisfiability", *Design, Automation and Test in Europe Conference*, 2001, pp.398-402.

[6] L. Lingappan, S. Ravi, N. K. Jha, "Test Generation for Non-Separable RTL Controller-Datapath Circuits Using a Satisfiability Based Approach", *IEEE International Conference on Computer Design*, 2003, pp.187-193.

[7] F. Fallah, S. Devadas, and K. Keutzer, "Functional Vector Generation for HDL Models Using Linear Programming and Boolean Satisfiability", *IEEE Trans. Computer-Aided Design*, vol.20, no.8, pp.994-1002, 2001.

[8] C. Y. Huang, K. T. Cheng, "Using Word-level ATPG and Modular Arithmetic Constraint-Solving Techniques for Assertion Property Checking", *IEEE Trans. Computer-Aided Design*, vol.20, no.3, pp.381-391, 2001.

[9] G. Parthasarathy, M. K. Iyer, K. T. Cheng, Li-C Wang, "An Efficient Finite-Domain Constraint Solver for Circuits", *ACM/IEEE Design Automation Conference*, 2004, pp.212-217.

[10] Shujun Deng, Jinian Bian, Weimin Wu, Xiaoqing Yang, Yanni Zhao, "EHSAT: An Efficient RTL Satisfiability Solver Using an Extended DPLL Procedure", *ACM/IEEE Design Automation Conference*, 2007, pp.588-593.

[11] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver", *ACM/IEEE Design Automation Conference*, 2001, pp.530-535.

[12] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A New Search Algorithm for Satisfiability", *International Conference on Computer Aided Design*, 1997, pp.220-227.