A REGISTER-TRANSFER LEVEL TESTABILITY ANALYZER

Yen-An Chen, Chun-Yao Wang, Ching-Yi Huang, and Hsiu-Yi Lin

Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan, R.O.C.

ABSTRACT

This paper presents a statistic-based method to estimate the testability of a design at Register-Transfer Level. This testability estimation technique is composed of a new proposed high-level design representation and a Monte Carlo simulation which exploits a statistic model to bound the error rate and confidence level of simulation results. The experimental results show that the proposed method can efficiently report more than 60% hard-to-test points of an RL design on average prior to the synthesis task.

I. INTRODUCTION

As designs are getting larger and more complicated, design for testability (DFT) techniques are getting more important to testing. This is because DFT techniques can elevate the testability of hard-to-test points of a circuit after manufacturing process. In general, designers need to know the locations of those hard-to-test points in a design before applying the DFT techniques. These hard-to-test points, however, are usually determined by gate-level testability analysis tools. Thus, designers either apply DFT techniques at gate-level, or re-implement the designs at Register-Transfer Level (RTL) for improving the testability of those hard-to-test points. However, the overhead of applying DFT techniques at gate-level might be large. On the other hand, designers usually question about whether or not these hard-to-test points are eliminated if they re-implement the design at RTL. Thus, to earlier locate the low testability point, we propose a testability measurement technique at RTL. It provides the testability information prior to gate-level netlists and could reduce the design costs and shorten the design cycle.

This work was supported in part by the National Science Council of Taiwan under grants NSC 99-2628-E-007-096, NSC 99-2220-E-007-003, and NSC 100-2628-E-007-031-MY3

There is much research estimating testability of a gate-level design in the last three decades. [1], [2], [4]-[7], [9], [11], [13]-[14], [20]. However, only a few papers that address the testability estimation at RTL [8], [12], [15], [17]-[19]. Logical Virtual Prototyping method [8], which utilizes a fast synthesis engine with a generic library is proposed recently. The generic library is composed of AND, OR, and NOT gates and is used to form a virtual gate-level hierarchical netlist. Then it applies scan chain insertion and Automatic Test Pattern Generation (ATPG) techniques to estimate the fault coverage rather than testability. However, we have known that the fault coverage is strongly associated with a test pattern set. A lower fault coverage only implies the test set is not good enough, but not indicates that the design contains many hard-to-test points. Thus, the approach needs an ATPG tool to precisely calculate the fault coverage. Furthermore, in general, synthesizing RTL description into generic gate-level netlist and applying ATPG to the netlist for fault coverage are both time-consuming.

FSTAFAN [12] calculates fault detection probability by applying a well-known testability analysis algorithm, STAFAN [9]. It only uses the fault-free simulation to evaluate the testability, and hence gaining the benefit of efficiency over other testability analysis methods. However, the FSTAFAN algorithm applies exhaustive vectors to a module to evaluate its fault coverage under a high-level design description. Hence, this approach is not practical to a module with a large amount of inputs.

Instead of pattern simulation, Strnadel proposed a graph-searching algorithm for estimating the testability of RTL designs [15]. It proposed some formulae and rules to compute and propagate controllability and observability, and then obtained the number of controllable or observable nodes in a design. Finally, it estimated the testability of the design by the number of those controllable and observable

978-1-4577-1617-1/11/$26.00 ©2011 IEEE

nodes. The algorithm may be efficient because of vectorless. However, the estimated testability might be inaccurate when the design contains lots of high-level constructs. This is the constraint of this work.

In this work, we propose a statistic-based method to estimate the testability of a design at RTL. To accomplish this, we propose a new representation *Register-Transfer Level with Assignment Decision* (RTL-AD), which is similar to Assignment Decision Diagram (ADD) [3], but is more appropriate to precisely computing the testability and easily performing parallel simulation. Furthermore, our approach applies a statistic method in the Monte Carlo Simulation to bound the error rate and confidence level. As a result, our approach can directly report the testability of each assignment statement in an RTL design.

II. RTL-AD STRUCTURE

A. *The Skeleton of RTL-AD*

RTL-AD contains six components as shown in Fig. 1: *data node*, *assignment decision node*, *assignment selection node*, *statement flow node*, *logical node,* and *arithmetic node*. The data node can be either a read node or a write node. The assignment decision node decides whether or not the data source can be passed to the target. The assignment selection node has multiple data sources at a time and decides which data source could be passed through. The statement flow node determines the switching timing of all assignment decision nodes and assignment selection nodes. All operators are divided into logical node, e.g., & (AND), | (OR), or ^ (XOR), and arithmetic node, e.g., + (addition), $\gg$ (right shift), or % (modulo). In other words, logical nodes are the same as logic gates, and arithmetic nodes have the functions of high-level arithmetic operations.

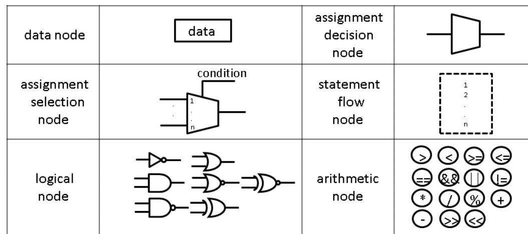

Figure 1: The six components of RTL-AD structure.

B. Transformation of RTL-AD from RTL Description

To better explain the construction of RTL-AD, we use a hierarchical design as shown in Fig. 2 to go through the construction of RTL-AD, which is very efficient. There are five steps to construct the RTL-AD from an RTL description.

B.1 Generate statement flow

The statement flow acts as an FSM to control the simulation sequence of these assignment statements by controlling the corresponding assignment decision nodes and assignment selection nodes. Before generating the statement flow, we first label each assignment statement a unique ID as shown in Fig. 2.

In general, the assignment statements within an *always* block are sequentially executed. However, for the assignment statements that belong to the same high-level construct, e.g., *if-else* block, *case* block, they are parallel executed. For example in Fig. 2, since the assignment statements *3*, *6*, *7* of module A belong to the same if-else block, they are parallel executed. Thus, the initial statement flow of module *A* is shown in Fig. 3(a). Since the left-hand side of assignment statement *7* is *Dout*, which is also the left-hand side of the assignment statements *4* and *5* which are controlled by variables *con* and *DFF*. Therefore, the assignment statement *7* is postponed to be executed in the same level of the assignment statements *4* and *5* as shown in Fig. 3(b).

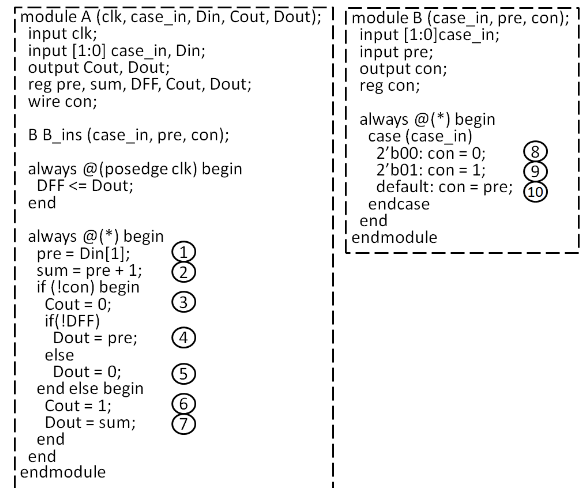

Figure 2: An example of hierarchical RTL design.

B.2 Build data path and condition flow

The data path is a passage for data transfer. The condition flow includes the control information in all conditional statements. We insert assignment decision nodes and assignment selection nodes in between the left-hand side and right-hand side of all assignment statements in building the data path. Afterward, we build the condition flow by connecting

the control variables of the condition statement to the corresponding assignment decision nodes as shown in Fig. 4.

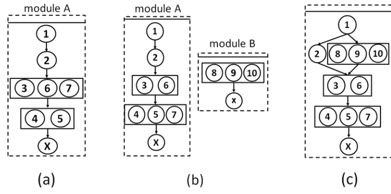

Figure 3: (a) The initial statement flow of module A. (b) The statement flow of the example in Fig. 2. (c) The complete statement.

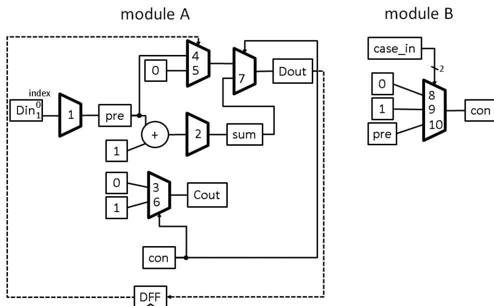

Figure 4: The data path and condition flow.

B.3 Combine all modules

According to the module instantiation in this example, the connection ports, *case_in*, *pre*, and *con,* are connected together. Hence, the data path and condition flow of module *A* and module *B* are combined into one complete structure as shown in Fig. 5.

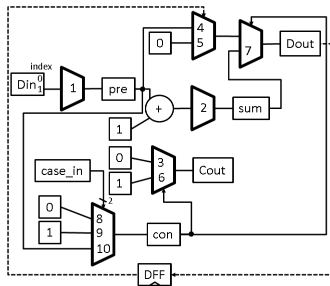

Figure 5: The complete data path and condition flow.

The statement flows of all modules also have to be combined. In this example, module *B* communicates with module *A* through variables *pre* and *con*, the statement flow of module *B* is inserted between the assignment statements *1* and *3_6* as shown in Fig. 3 (c).

B.4 Accommodate the full scan chain

After inserting the full scan chain, each flip-flop is replaced by a pair of pseudo primary input (PPI) and pseudo primary output (PPO). Hence, we can apply vectors at PPIs and observe the results at PPOs.

B.5 Connect the data path and statement flow

All assignment decision and assignment selection nodes are connected to its corresponding nodes in the statement flow as shown in Fig. 6. Therefore, the simulation procedure can be successfully controlled by the statement flow.

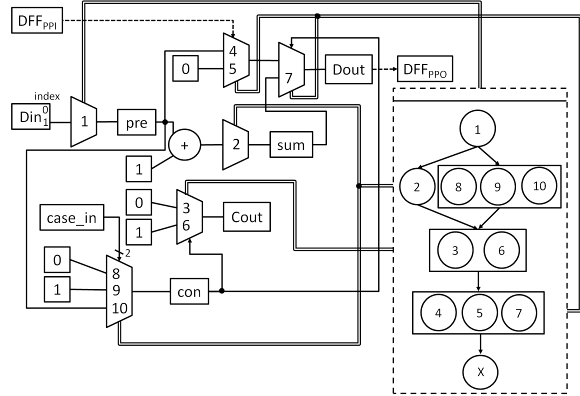

Figure 6: The complete RTL-AD structure.

III. MONTE CARLO METHOD

A. The Components of Monte Carlo Method

There are four major components in Monte Carlo method, including *Random Simulation*, *Sampling Rule*, *Scoring,* and *Error Estimation*.

A.1 Random Simulation

To facilitate a high-level parallel simulation mechanism, which was not proposed in previous works, we include three types of simulation methods, *logic simulation*, *virtual simulation*, and *high-level simulation*, in our approach.

Logic Simulation This type of simulation is employed for all logical nodes. It uses the same concept of word-level parallel simulation used in gate-level netlist.

Virtual Simulation This type of simulation is applied for the arithmetic nodes except for the division and modulo operations. We use an example of comparison operation as shown in Fig. 7(a) to demonstrate this idea. Fig. 7(b) shows its corresponding RTL-AD structure and the generated random patterns for the variable *con*. The constant *2* is represented as *10* where the Most Significant Bit (MSB)

is indicated by the index *1*. We use a comparator as shown in Fig. 7(c) to do the comparison operation to facilitate the parallel simulation.

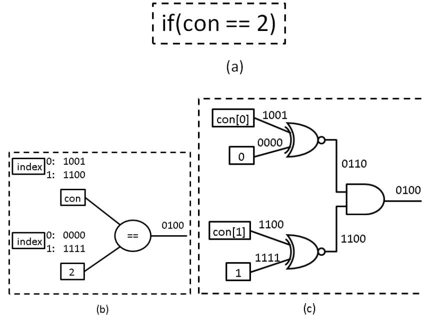

Figure 7: An example for virtual simulation.

High-level Simulation For the complex arithmetic operations, such as division and modulo, we implement them from the high-level viewpoint. We use an example of division operation *A/B* as shown in Fig. 8 to demonstrate this idea. We first convert the random values of *A* and *B* from binary to decimal, and then repeat the division operation four times to get the answers *(3, 1, 0, 0)*. The computed results are converted back from decimal to binary again, and are saved in the data node *C*.

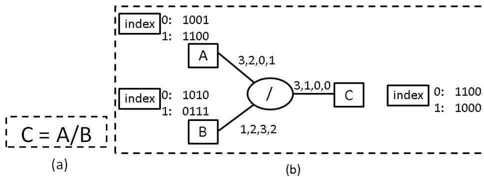

Figure 8: An example for high-level simulation.

A.2 Sampling Rule

Each sampling in Monte Carlo method will return a result for the desired answer, which is the fault detection probability (testability) of each wire in the RTL-AD structure in this work. Here, we apply the similar method in [6] to obtain this value. Due to page limit, we skip the detail of this part.

A.3 Scoring

With the sampling rule, the data obtained from a sufficient amount of samplings are collected to approximate the exact result.

The next important issue we have to deal with is how many samplings are sufficient for just obtaining an accurate result. Specifically, when considering parallel simulation in this work, the bit width of one simulation also relates to the number of samplings needed. That is, when the bit width is larger, the number of samplings can be fewer, and vice versa. On the other hand, since our statistic model uses the t-*distribution* [10], which is more appropriate for the sampling times under 30, to estimate the error rate and confidence level. We also take this issue into consideration in determining the bit width r in our parallel simulation architecture when conducting the experiments.

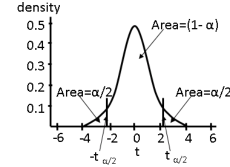

Figure 9: A demonstration of confidence level.

A.4 Error Estimation

We exploit the *Confidence Interval* in statistics to estimate the error rate of fault detection probability of a wire as compared with the mean of the normal distribution. Suppose that we have N samplings of the fault detection probability of a wire w, then we can compute the sample standard deviation of this wire denoted as $sd(w)$ among the samplings. The parameter of the predefined confidence level is α as shown in Fig. 9 where $(1 - \alpha) \times 100\%$ represents the confidence level. Then we can look up the value $t_{\frac{\alpha}{2}}$ from the t-distribution table with $(N - 1)$ degrees of freedom. Hence, we have $(1 - \alpha) \times 100\%$ confidence level that the estimated error rate of the fault detection probability of w, $e(w)$, is expressed as EQ(1) [10].

$$e(w) = \frac{t_{\frac{\alpha}{2}} \times sd(w)}{\sqrt{N}} \quad (1)$$

Consequently, for a desired error rate ϵ in the fault detection probability estimation, and for a given confidence level $(1 - \alpha) \times 100\%$, we have to repeat sampling until

$$\frac{t_{\frac{\alpha}{2}} \times sd(w)}{\sqrt{N}} < \epsilon \quad (2)$$

EQ(2) is called the *stopping condition* of our Monte Carlo approach. However, it is time-consuming to check whether the fault detection probability of every wire meets the stopping condition defined in EQ(2). As a result, a heuristic of selecting some appropriate check points such that most wires would satisfy the stopping condition while the selected check point wires did is proposed. In addition, after observing EQ(2), we realize that only the sample standard deviation $sd(w)$ of each wire differs from others. Thus, we use the $sd(w)$ to represent each wire and choose the top 10% of wires with higher $sd(w)$ as the check points after several initial sam-

plings. When all those check points meet the stopping condition, the sampling is terminated.

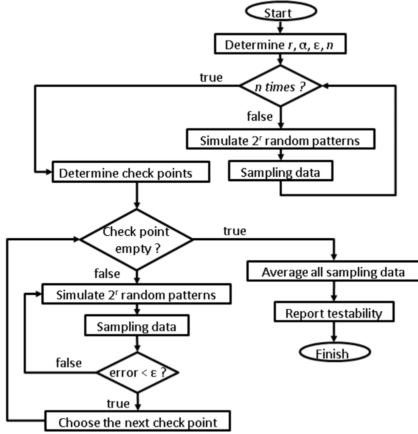

Figure 10: The flow chart.

B. Monte Carlo Flow

The flow chart of Monte Carlo method is shown in Fig. 10. First, the parameter r of RPG, confidence level α, error rate ϵ, and the number of initial sampling *n* are specified. After *n* samplings, we identify the check points. If a check point has met the stopping condition, we proceed for the next check point. When all the check points have been processed, we average the results of all the samplings and report the fault detection probability of each assignment statement.

IV. EXPERIMENTAL RESULTS

A. Signature-based Verification Method

Our approach can directly indicate the testability of each assignment statement in an RTL design. To show the accuracy of the repeated results, we compare the results to that obtained by the corresponding gate level circuit using the proposed signature-based verification method described as follows. We launch the simulation on the RTL-AD structure and the corresponding gate-level netlist of a benchmark. If the signature on an RTL-AD node is the same as that of one and only one gate-level node, this gate-level node is regarded as a *matching point*. The matching nodes are used to compare with the exact result in error rate analysis.

B. Experimental Results

The experiments are conducted over a set of RTL ISCAS'89 benchmarks and some practical designs to evaluate the performance and accuracy of our approach on a 3.0GHz Linux platform. We compute the testability of each assignment statement based on the proposed RTL-AD structure. The gate-level netlists in our experiments are obtained by Design Compiler [16] using the TSMC 0.18μm library.

In the experiments, we set the parameters as r = 13, α= 0.001, ε= 0.005 and *n* = 10, which means that our results will allow having the error rate of 0.005 with 99.9% confidence level. Then, we apply the Monte Carlo method with those parameters to the RTL-AD structure. We also compute the exact testability by BDD as [5] did for comparison. The experimental results are summarized in two tables. Table 1 is to show the efficiency and the error rate distribution as compared with the exact value of our approach in reporting the estimated testability.

Table 1: Error rate distribution of our approach.

circuit	RTL-AD		error rate (%)					
	| nodes |	exe	0~5	10	20	30	50	50↑
s27	46	0.07	50	0	34	0	8	8
s208	370	0.55	51	9	3	8	23	6
s298	464	0.8	34	8	8	22	21	7
* s344	572	0.97	30	11	19	10	17	13
* s349	578	0.81	31	9	18	10	20	12
s382	626	1.01	43	7	18	17	5	10
s386	538	0.66	84	0	5	0	1	10
s400	651	0.72	43	6	15	18	7	11
* s420	768	1.21	60	4	2	6	15	13
s444	713	1.22	44	9	17	16	6	8
* s510	847	1.62	54	8	9	7	12	10
* s526	792	1.24	42	8	11	22	12	5
* s641	1208	1.78	29	8	18	15	12	18
* s713	1280	1.68	30	7	18	15	11	19
* s820	1215	1.7	74	10	3	3	6	4
* s832	1222	1.96	74	10	3	3	7	3
* s838	1564	2.37	69	3	1	4	15	8
* s953	1498	2.29	63	3	1	1	5	27
* s1423	2241	3.5	44	5	12	26	11	2
s1488	2085	2.74	50	13	14	6	11	6
s1494	2072	2.97	52	12	13	7	11	5
* s5378	9438	16.85	43	4	11	16	13	13
*** s9234**	**17546**	**33.86**	**52**	**4**	**2**	**9**	**16**	**17**
* s13207	25430	43.23	45	1	4	7	21	22
* s15850	30623	55.75	46	3	8	6	20	17
** s35932	56963	92.65	61	9	13	15	2	0
** s38417	74119	149.21	45	2	6	11	20	16
** s38584	61686	113.45	40	5	12	18	14	10
counter8	17	0.06	100	0	0	0	0	0
vendor	219	0.99	100	0	0	0	0	0
blackjack	248	2.4	88	0	6	0	0	6
* rankf	1090	13.81	100	0	0	0	0	0
div8	2893	24.3	83	7	8	0	0	2
average	-	***17.5***	***56***	***6***	***10***	***9***	***10***	***9***

In Table 1, Column 1 shows the name of each benchmark. The next two columns show the number of nodes in RTL-AD structure and the CPU time for reporting the results. Columns *4~9* show the error rate distribution, as compared with the exact value. For example, Column 0~5 shows the percentage of matching points having 0~5% error rate as compared to the exact results when applying Monte Carlo method on RTL-AD. Column 20 shows the percentage of matching points having 10%~20% error rate as compared to the exact results.

The circuits marked with "*" and "**" mean that they cannot be represented by BDD approach. Thus, the results of these circuits are approximated by the simulation of a large amount of patterns, specifically

2^{20} and 2^{15} in our experiments, respectively. We only report the faults of those matching points by using the signature-based verification method mentioned. For example in *s9234* benchmark, it has 7546 nodes, total CPU time is 33.86 seconds, and 52% of matching points are within 0~5% error rate.

According to Table 1, the CPU time required for the proposed approach is not much which indicates that the RTL-AD construction is not a burden for the approach. Furthermore, 56% of matching points in a benchmark are within the error rate range of 0~5% on average. Thus, the proposed statistic approach is quite accurate.

Table 2: The results of low testability point identification.

circuit	| node having low testability |		ratio
	ours	exact	
s27	5	7	71%
s208	55	98	56%
s298	33	83	40%
s344	65	126	52%
s349	51	108	47%
s382	67	114	59%
s386	63	74	85%
s400	55	103	53%
s420	121	200	51%
s444	62	103	60%
s510	84	128	66%
s526	71	141	50%
s641	162	353	46%
s713	183	389	47%
s820	118	139	85%
s832	**123**	**145**	**85%**
s838	192	276	70%
s953	225	348	65%
s1423	181	386	47%
s1488	188	260	72%
s1494	221	302	73%
s5378	928	1858	50%
s9234	1593	3256	49%
s13207	2037	6333	32%
s15850	2663	6492	41%
s35932	1341	2220	60%
s38417	6073	17692	34%
s38584	7074	14539	49%
counter8	0	0	-
vendor	3	3	100%
blackjack	8	9	89%
rankf	0	0	-
div8	206	212	97%
average	-	-	***61%***

From the application viewpoint, we also would like to know if our approach can identify those low testability statements in RTL designs. Table 2 shows the comparison between our approach and exact approach on identifying the low testability statements. A point with less than 20% testability is regarded as a low testability point in this paper. Column 2 shows the number of low testability points identified by our approach. Column 3 is the exact number of low testability points from netlists. The last column shows the ratio of identified low testability points as compared to the exact number. Take *s832* as an example, 145 low testability points are identified by the exact method, and 123 or 85%, low testability points are identified by our approach. According to Table 2, 61% of low testability points are identified on average by our approach.

V. CONCLUSION

In this paper, we propose a statistic-based testability analyzer based on the proposed RTL-AD structure and Monte Carlo method at RTL design description. It directly reports the testability of each assignment statement in RTL designs. With this approach, designers can efficiently identify many low testability statements such that the design cycle can be shortened when considering design-for-testability issue.

REFERENCES

1. V. D. Agrawal and S. C. Seth, “Probabilistic testability,” *in Proc. ICCD*, pp. 562–565, 1985.
2. F. Brglez, “On testability of combinational networks,” *in Proc. ISCAS*, pp. 221–225, 1984.
3. V. Chaiyakul and D. D. Gajski, “Assignment decision diagram for high-level synthesis,” UC Irvine, Technical Report ICS-TR-92-103, Dec. 1992.
4. S. Chakravarty and H. Hunt, “On computing signal probability and detection probability of stuck-at faults,” *IEEE TC*, pp. 1369–1377, Nov. 1990.
5. S. C. Chang, W. B. Jone, and S. S. Chang, “Tair: Testability analysis by implication reasoning,” *IEEE TCAD*, pp. 152–160, Jan. 2000.
6. C.-C. Chiou, C.-Y. Wang, and Y.-C. Chen, “A statistic-based approach to testability analysis,” *in Proc. ISQED*, pp. 267–270, 2008.
7. L. H. Goldstein and E. L. Thigpen, “Scoap: Sandia controllability/observability analysis program,” *in Proc. DAC*, pp. 190–196, 1980.
8. Y. Huang, N. Mukherjee, W.-T. Cheng, and G. Aldrich, “A rtl testability analyzer based on logical virtual prototyping,” *in Proc. ATS*, pp. 121–124, 2007.
9. S. K. Jain and V. D. Agrawal, “Statistical fault analysis,” *IEEE Design & Test of Computers*, vol. 2, pp. 38–44, Feb. 1985.
10. I. R. Miller, J. E.Freund, and R. Johnson, “Probability and statistics for engineers,” *Englewood Cliffs, NJ: Prentice Hall*, 1990.
11. K. P. Parker and E. J. McCluskey, “Probabilistic treatment of general combinational networks,” *IEEE TC*, pp. 668–670, 1975.
12. C. P. Ravikumar and G. S. Saund, “A stafan-like functional testability measure for register-level circuits,” *in Proc. ATS*, pp. 192–198, 1995.
13. J. Savir, G. S. Ditlow, and P. H. Bardell, “Random pattern testability,” *IEEE TC*, pp. 79–90, Jan. 1984.
14. S. C. Seth, L. Pan, and V. D. Agrawal, “Predict: Probabilistic estimation of digital circuit testability,” *in Proc. FTCS*, pp. 220–225, 1985.
15. J. Strnadel, “Testability analysis and improvements of register-transfer level digital circuits,” *Computing and Informatics*, pp. 1001–1024, Sep. 2006.
16. SYNOPSYS. [Online]. Available: http://www.synopsys.com
17. M. Takahashi, R. Sakurai, H. Noda, and T. Kambe, “A testability analysis method for register-transfer level descriptions,” *in Proc. ASP-DAC*, pp. 307–312, 1997.
18. D. T. Wang, “An algorithm for the generation of test sets for combinational logic network,” *IEEE TC*, pp. 742–746, July 1975.
19. S.-J. Wang and T.-H. Yeh, “High-level test synthesis with hierarchical test generation for delay-fault testability,” *IEEE TCAD*, pp. 1583–1596, Oct. 2009.
20. S.-C. Wu, C.-Y. Wang, and Y.-C. Chen, “Novel probabilistic combinational equivalence checking”, *IEEE TVLSI*, pp.365-375, April 2008.