New Trends in RTL Verification: Bug Localization, Scan-Chain-Based Methodology, GA-Based Test Generation

Khaled Salah
Mentor Graphics
Cairo, Egypt
Khaled_mohamed@mentor.com

Abstract- In this paper, New Trends in RTL Verification is introduced, this includes: bug localization, Scan-Chain-Based methodology, and GA-Based test generation. Manual bug localization in Hardware Description Language (HDL) designs is a time-consuming process; therefore, there is an increased demand for automated techniques that can speed-up the bug localization process. In this paper, a code coverage-based method is proposed to rank suspicious code according to its probability of containing bugs which may result in significantly reducing the debugging time by starting with the first candidate bug location. This method is different from formal methods for error localization such as assertion-based methods which are not suitable for large designs as this method is a simulation-based technique and suitable for large designs. Results show that our method can detect errors in large designs up to several thousand lines of RTL code in few minutes with high accuracy compared to time consumed in hours using manual bug localization with success rate of 100%.An online RTL-level scan chain methodology is proposed to reduce debugging time, effort and accelerate IP emulation. Run-time changes of the values of the signals of the IP during execution-time can be done by the proposed scan-chain methodology. A utility tool was developed to help ease this process. Our experiment shows that, the area overhead is neglected compared to the gained performance benefits. But, design requires more compilation time. The main challenge in using constraint random testing (CRT) is that manual analysis for the coverage report is needed to find the untested scenarios and modify the test cases to achieve 100% coverage. We need to replace the manual effort by an automatic method or a tool that will be able to extract the coverage report, identify the untested scenarios, add new constraints, and iterate this process until 100% coverage is attained. In other words, we need an automated technique to automate the feedback from coverage report analysis to test generation process. In this paper, the implementation of this automatic feedback loop is presented. The automatic feedback loop is based on artificial intelligence technique called genetic algorithm (GA). This technique accelerates coverage-driven functional verification and achieves coverage closure rapidly by covering uncovered scenarios in the coverage report.

Keywords—Trends; Localization; Bug; Genetic Algorithm; Scan-Chain.

I. INTRODUCTION

Functional verification is a required process to ensure that the design is in accordance with the specification. Due to the increasing design complexity of SoC systems, the cost of functional verification has significantly increased. According to ITRS, [1]-[2], verification process is now considered a bottleneck as it consumes up to 70% of the design cost. In order to keep the production cost low, it is required to detect bugs as soon as possible. This work targets localization of functional errors. While there a lot of verification methodologies for error detection in RTL design, there is fewer work for debugging the error which includes localization and correction stages. Moreover, most of related works are concentrating on gate-level error localization, [3]-[5], and are applied to small designs.

Here, we are focusing on the RTL-level and large designs. Detecting and locating the source of erroneous behavior in large and complex RTL design is challenging. In this paper, we present a novel approach for bug localization methodology to address this challenge using information from regression suit results about failed and passed testcases and number of statements executed by each test. The idea is inherited from software domain [6]-[8]. Moreover, we present a proof of concept for this idea using a Verilog-based case studies.

Simulation-based verification scheme of large sophisticated intellectual property (IPs) is considered a time consuming process. Mainly, there are two famous methods to help accelerate simulation process and reduce verification time: hardware acceleration, and hardware RTL emulation. The RTL hardware accelerator solutions are based on using application-specific ASICs, each contains special-application processors and memories [9]-[12]. The RTL hardware emulators are based on using FPGAs, where the design is synthesized into a gate-level netlist. However, most hardware emulator does not provide easy debugging capability at run-time. In this paper, a scan-chain scheme is proposed to reduce debugging time. The proposed

method provides internal glue-block which automatically replaces any signal with a mux and extra input, so that at run time if we enable this method we can replace any internal signal by a forced one to use it for debugging.

The efficiency of the verification process is proportional to achieving the coverage goals in less simulation time. Many different test data generation methods like random test data generator have been proposed in the literature [13]-[15]. In this wok, the verification process problems will be considered as an optimization problem and GA is proposed as a novel method for test generation to help reaching 100% functional coverage in less time than conventional methods.

The rest of this chapter is organized as follows. In Section II, bug localization scheme is presented. In Section III, scan-chain based methodology is introduced. In Section IV, GA-based test generation scheme is presented. Conclusions are given in Section V.

II. RTL BUG LOCALIZATION

In this section, the proposed methodology for RTL bug localization is given in Section II. A. Results are discussed in Section II.B.

A. *Proposed Methodology*

Given a set of statements (*S*) for which an HDL design exhibits an incorrect behavior, the objective of design debugging is to find the highly candidate statement that may be responsible for this incorrect behavior. The failing and passing testcases are used to find the bug location. If a statement is executed by more than two failing testcases, so this statement is more likely to have the bug. So, run the complete regression suite until the coverage is 100%, then extract the needed information about the passed and failed testcases and obtain a list of design statements executed by each test. The proposed methodology is shown in Fig. 1. An example to show how our proposed method works is shown in TABLE I, where we assume that our DUT has only one bug due to only one incorrect statement and we have 10 test-cases to test its behavior.

From TABLE I, the left columns shows how each RTL statement is executed by each testcase either it is failing or passing. An entry 1 indicates that the statement is executed by the corresponding test case and an entry 0 means it is not executed. The most right column shows the execution result with an entry 1 for a failed testcase and an entry 0 for a passing testcase. If a statement is executed by a successful test case, its likelihood of containing a bug is reduced.

The weighted probability is used to indicate that more successful executions imply less likely to contain the bug. So, the suspiciousness of each statement = the number of failed tests that execute it / the number of successful tests. And we will choose the maximum value to start with, *i.e*, the large rank. The proposed methodology for automated bug localization is shown in Fig.3.

B. *Experimental Results*

Experimental results show that the proposed method can detect errors in large designs up to several thousand lines of RTL code in few minutes with high accuracy compared to time consumed in hours using manual bug localization. Here, we only localize the error not correcting it. Other experiments are done on more bugs to observe the effectiveness of our methodology. We insert errors into some other parts of the code for the complex RTL design then we applied our methodology to locate the error. TABLE II reveals some results, where it is clear that our methodology reduces the time needed to localize the bug significantly. The effectiveness of this methodology varies for different designs, bugs, and testcases. Here, we assume that we have a rich and correct testcases.

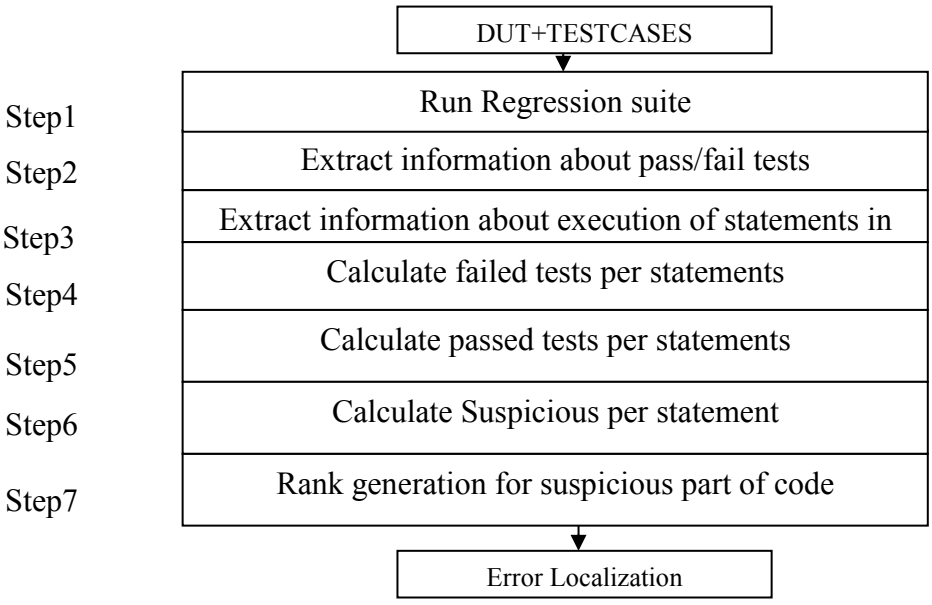


Fig. 1 The proposed methodology for automated bug localization.

TABLE I
A MOTIVATIONAL EXAMPLE TO DESCRIBE THE PROPOSED METHODOLOGY.

Test Suite	STATEMENTS															TEST STATUS (T)
	S0	S1	S2	S3	S4	S5	S6	S6	S7	S8	S9	S10	S11	S12	S13	P(0)/F(1)
Test1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Test2	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0
Test3	1	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0
Test4	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0
Test5	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0
Test6	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1
Test7	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1
Test8	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0
Test9	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1
Test10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Failed tests per statements	4	4	4	4	4	4	4	4	4	4	4	4	1	1	1	
Passed tests per statements	6	6	6	6	6	6	5	5	3	0	0	0	5	5	3	
Suspicious per statement	-2	-2	-2	-2	-2	-2	-1	-1	1	4	4	4	-4	-4	-2	
Weighted Suspicious per statement	0.67	0.67	0.67	0.67	0.67	0.67	0.8	0.8	1.33	Inf	Inf	Inf	0.2	0.2	0.33	

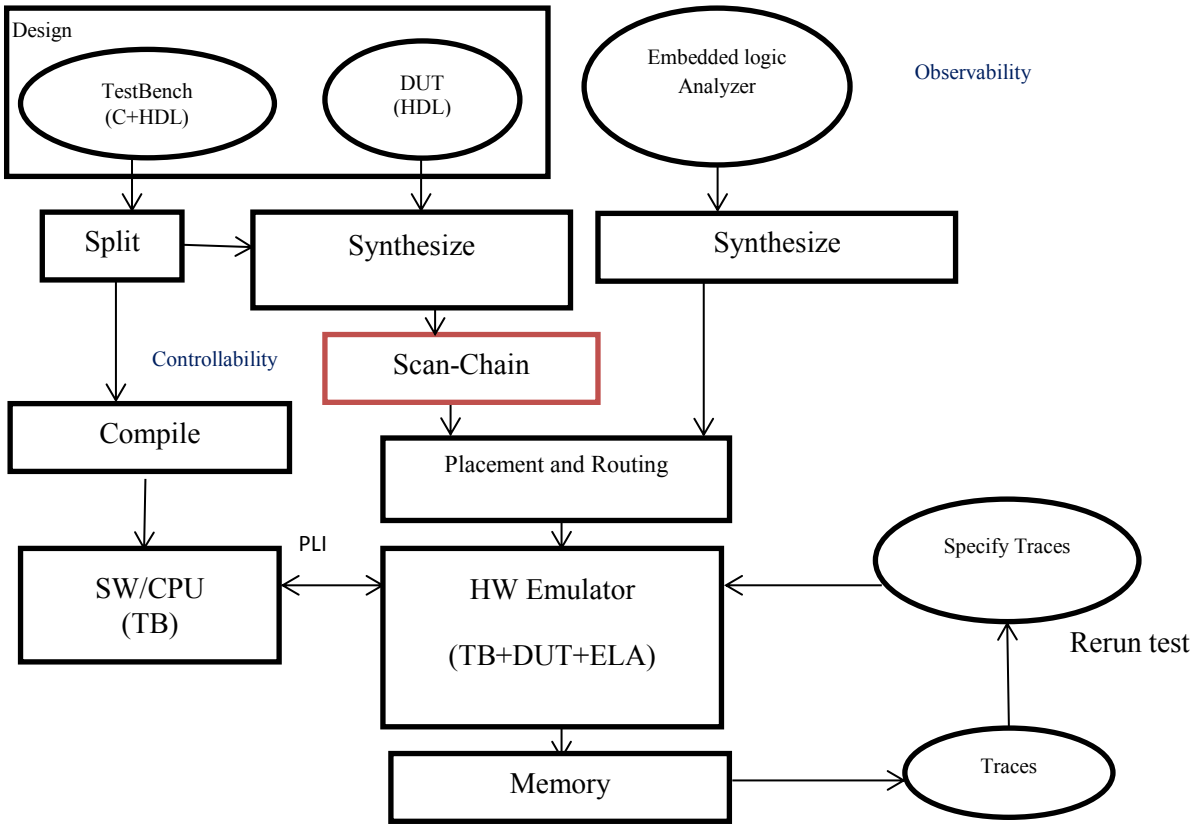
TABLE II
BUG LOCALIZATION TIME USING THE PROPOSED METHODOLOGY VERSUS MANUAL DEBUG IN A COMPLEX DESIGN, WHICH CONTAINS MORE THAN 5000 LINE OF RTL CODE

Bug ID	Time		Wrong behavior	Correct behavior
	The proposed methodology (min)	Manual Debug (Hours)		
Bug 1	3	1	x1=x2+x3+x4+x5	x1=x2-x3+x4+x5
Bug 2	4	2	If (y1)	If (~y1)
Bug 3	7	2	If (~y2)	If (y2)
Bug 4	5	2	cnt=cnt+2;	cnt=cnt+3;
Bug 5	2	1	if (~btst_card_en & ~strt_cmd_data_dly)	if (~btst_card_en & strt_cmd_data_dly)
Bug 6	3	3	TST_DATA <= 8'h00;	TST_DATA <= 8'h01;
Bug 7	10	2	MFSM_BUS <=MFSM_BUS_REG;	MFSM_BUS <=MFSM_BUS_REG/2;
Bug 8	6	2	if (CMD6_ARG [31] ==1)	if (CMD6_ARG [31] ==0)
Bug 9	7	1	current_state <= Data	current_state <= Rcv;
Bug 10	4	3	MFSM_OUT_ENABLE <= 4'hf;	MFSM_OUT_ENABLE <= 4'he;
Bug 11	2	1	If (OP_MODE ==2)	If (OP_MODE ==1)
Bug 12	9	1	MFSM_STRT_DATA_P2S <= 1'b1;	MFSM_STRT_DATA_P2S <= 1'b0;
Bug 13	4	2	bus_width_prev<= MFSM_WIDTH;	bus_width_prev<= MFSM_WIDTH/2;
Bug 14	5	1	MFSM_BUS_WIDTH <= 4'h0;	MFSM_BUS_WIDTH <= 4'h1;
Bug 15	2	1	MFSM_LEN <= 32'h0;	MFSM_LEN <= 32'h200;
Bug 16	3	1	strt_cmd_data_dly <= 1'b0;	strt_cmd_data_dly <= 1'b1;
Bug 17	2	2	If ((1'b1<<WRITE_BL_LEN)+ 1'b1))	If ((1'b1<<WRITE_BL_LEN)- 1'b1))
Bug 18	2	1	If ((blk_dis == 1'h1)	If ((blk_dis == 1'h0)
Bug 19	5	1	crc_dis<=(cnt_crc==16-NC)? 1'h1:1'h1;	crc_dis<=(cnt_crc==16-NC)? 1'h0:1'h1;
Bug 20	3	2	if (blk_no1 != blk_count)	if (blk_no1 == blk_count)
Bug 21	2	1	W_OR_R <= 0 ;	W_OR_R <= 1 ;
Bug 22	1	1	If (blk_len_cmd16 < blk_len)	If (blk_len_cmd16 > blk_len)
Bug 23	3	2	cnt4 <= cnt4 + 1;	cnt4 <= cnt4 - 1;
Bug 24	1	3	else if (~incr_rd_user_addr)	else if (incr_rd_user_addr)
Bug 25	5	1	Else (WRITE_BLK_MISALIGN)	Else (~ WRITE_BLK_MISALIGN)
Bug 26	1	1	erase_start_addr<(ERASE_SIZE)*512	erase_start_addr <(ERASE_SIZE+1)*512
Bug 27	2	2	data_cnt_cmd25<= 32'h0;	data_cnt_cmd25<= 32'h1;
Bug 28	4	1	data_cnt_cmd25_en <= 1'b0;	data_cnt_cmd25_en <= 1'b1;
Bug 29	1	2	TST_DATA <= 8'h00;	TST_DATA <= 8'h80;

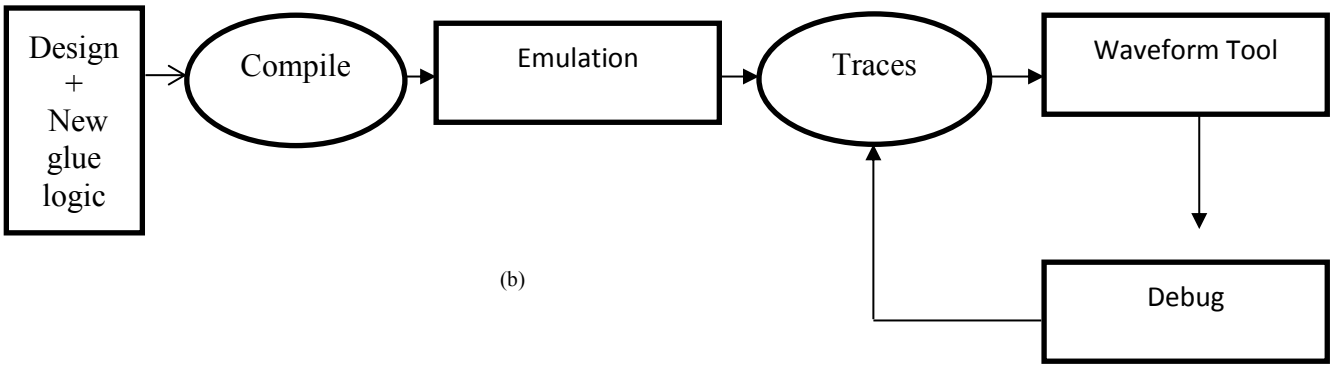
III. THE PROPOSED RTL-LEVEL SCAN-CHAIN METHODOLOGY

RTL simulation provides system on chip (SoC) verification with full debugging capabilities, but its disadvantages are the low-speed simulation for complicated RTL design. By using FPGA-based RTL emulation, we can have high-speed simulation. But, it is not easy to debug it because it has poor-capabilities visibility. Other solutions provide full debug capabilities such as RTL emulators, but the offline debugging method needs to recompile the whole design, which slows the verification process. In this paper, we propose an online RTL-level scan-chain-based methodology is proposed for accelerating IP emulation debugging time at Run-Time. This method provides internal glue-block which automatically replaces any signal with a mux and extra input, so that at run time if we enable this method we can replace any internal signal value by a forced one. Our experiment shows that, the area overhead is neglected compared to the gained performance benefits. The conventional emulation flow versus the proposed scan-chain based emulation flow is shown in Fig. 2 and Fig. 3 respectively.

To illustrate the proposed method, we assume the example shown in Fig. 3 (a), where: *out <= (A+B) * C*; where *C* is predetermined value that we want to change it at run-time, we compile the design and run emulation. If we want to change value of *C*, we have to recompile the whole design. Sometimes, it takes very large time depending on the complexity of the design. So, here we propose to use the online RTL-level scan-chain methodology to be able to change the value of *C* at run time without recompiling the whole design which accelerates the emulation debugging time. We will create a utility tool that instantiates glue logic and a mux with each "reg" definition in the RTL file, the glue logic is a null connection which puts the input into the output as depicted in Fig. 3(b). So, the designer can change the value of the signal at run time. It will be automatically auto-generated for all the registers defined in the design. Our experiment shows that, the area overhead is neglected compared to the gained performance benefits.



(a)



(b)

Fig. 2 Proposed Emulation Flow (online flow), synthesizable testbench methodology, scan-chain methodology, a) detailed, (b) simplified.

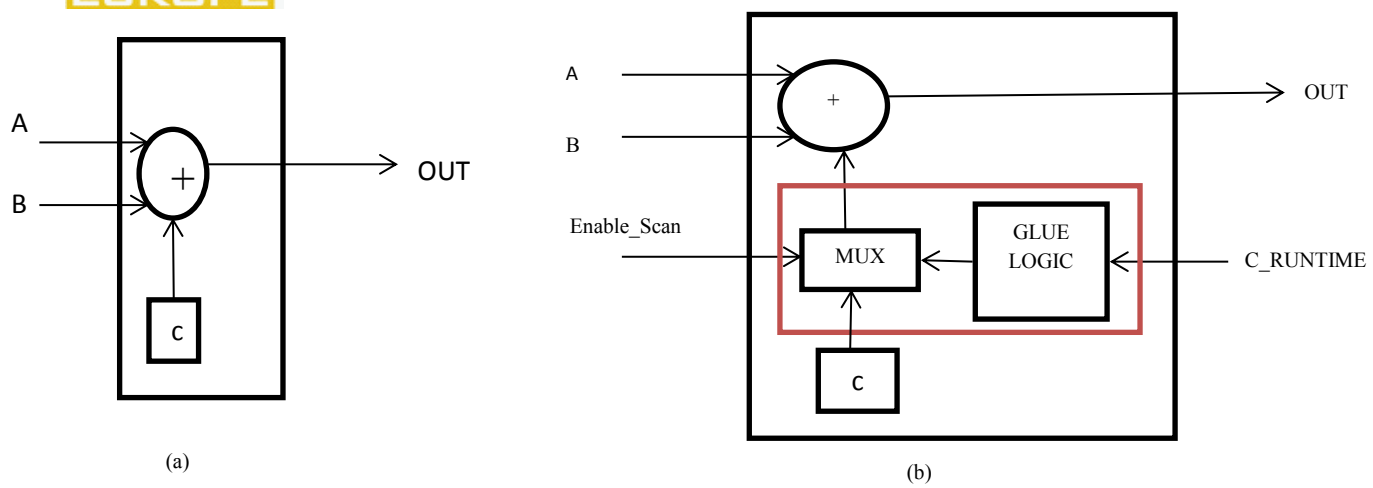


Fig. 3 (a) Normal design example, (b) proposed scan-chain methodology for the design example in (a).

IV. GA-BASED PROPOSED METHODOLOGY

GA is the heuristic (experience-based) search and time-efficient learning and optimization techniques that mimic the process of natural evolution based on Darwinian Paradigm. Thus genetic algorithms implement the optimization strategies by simulating evolution of species through natural selection. Each cell of a living thing contains chromosomes (strings of DNA), each chromosome contains a set of genes (blocks of DNA), and each gene determines some aspect of the organism (like eye color). In other words, parameters of the solution (genes) are concatenated to form a string (chromosome). In a chromosome, each gene controls a particular characteristic of the individual. The population evolves towards the optimal solution. Evolution based on "survival of the fittest". Genetic algorithms are well suited for hard problems where little is known about the underlying search space. So, it is considered a robust search and optimization mechanism.

The genetic algorithm used in this work consists of the following steps or operations, [13] -[19]:

1) Initialization and encoding:

The GA starts with the creation of random strings, which represent each member in the population.

2) Evaluation (Fitness):

The fitness used as a measure to reflect the degree of goodness of the individual, is calculated for each individual in the population:

3) Selection

In the selection process, individuals are chosen from the current population to enter a mating pool devoted to the creation of new individuals for the next generation such that the chance of a given individual to be selected to mate is proportional to its relative fitness. This means that best individuals receive more copies in subsequent generations so that their desirable traits may be passed onto their offspring. This step ensures that the overall quality of the population increases from one generation to the next.

4) Crossover :

Crossover provides the means by which valuable information is shared among the population. It combines the features of two parent individuals to form two children individuals that may have new patterns compared to those of their parents and plays a central role in Gas. The crossover operator takes two chromosomes and interchanges part of their genetic information to produce two new chromosomes.

5) Mutation:

Mutation is often introduced to guard against premature convergence. Generally, over a period of several generations, the gene pool tends to become more and more homogeneous. The purpose of mutation is to introduce occasional perturbations to the parameters to maintain genetic diversity within the population.

6) Replacement:

After generating the offspring's population through the application of the genetic operators to the parents 'population, the parents' population is totally replaced by the offspring's population. This is known as no overlapping, generational, replacement. This completes the "life cycle" of the population.

7) Termination

The GA is terminated when some convergence criterion is met. Possible convergence criteria are: the fitness of the best individual so far found exceeds a threshold value; the maximum number of generations is reached.

The proposed methodology to generate testcases using GA is as follows: generate stimulus based on the feedback from previously generated stimulus to cover areas which were not explored by previously applied tests. During each stimulus cycle,

coverage results are collected and sent as an input to the genetic algorithm and used as a guideline for next stimulus. The next stimulus will be more effective compared to randomly-generated one (Fig. 4). The fitness function here is chosen to maximize the functional coverage percentage, where:

$$Fitness = Functional\ Coverage\ ratio \quad (1)$$

Simulation results show that:

1) Coverage holes can be hit automatically with less effort and less time (Fig. 5).
2) Computational resources should be low.

The results for some designs are reported in TABLE III, where it is clear that using GA, we can reach 100% coverage in less time will less number of stimulus.

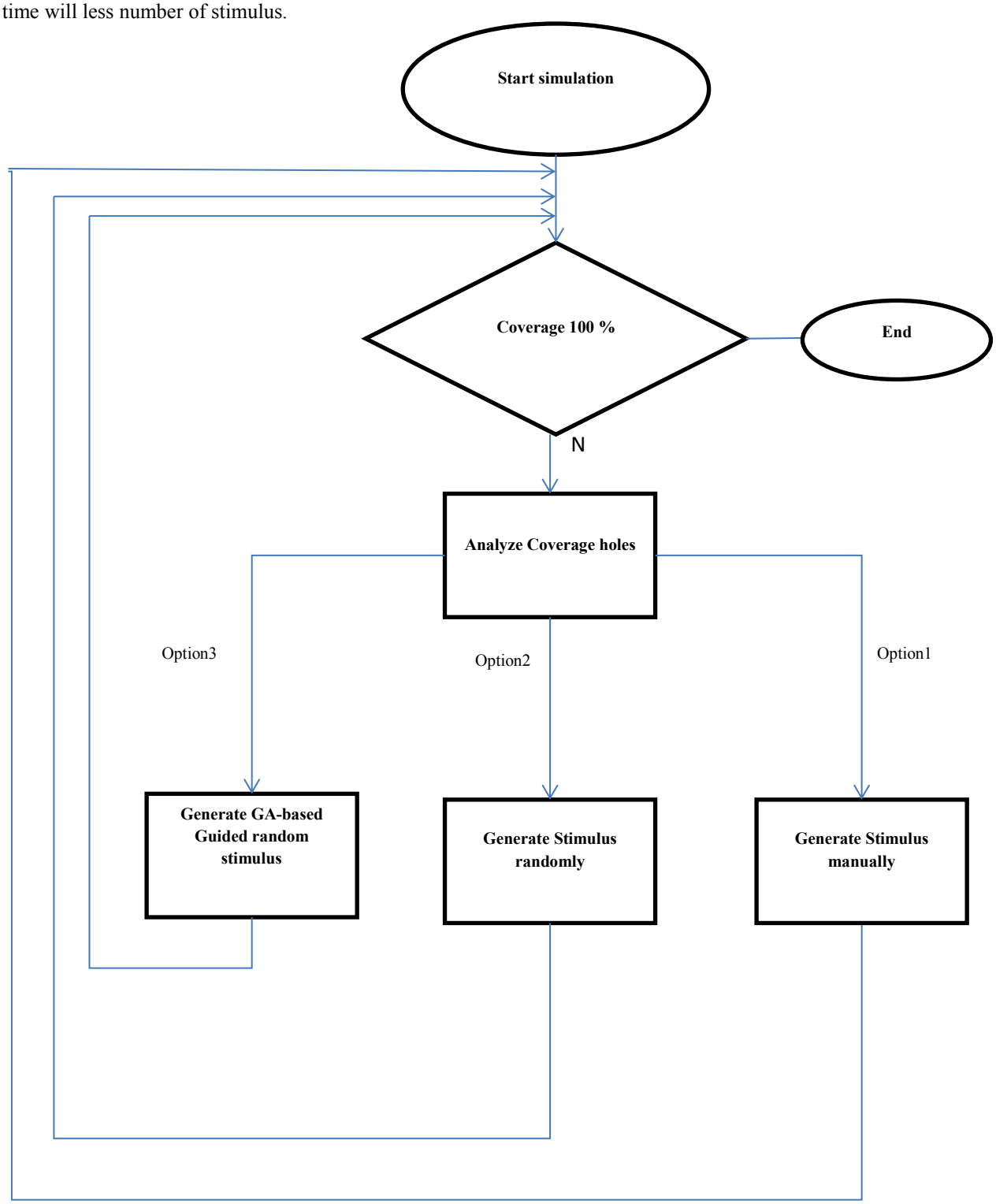


Fig. 4 The proposed GA methodology to speedup coverage closure. Using genetic algorithms, there is no test redundancy.

Number of generations =200, population size =50, type of crossover =Multi-point

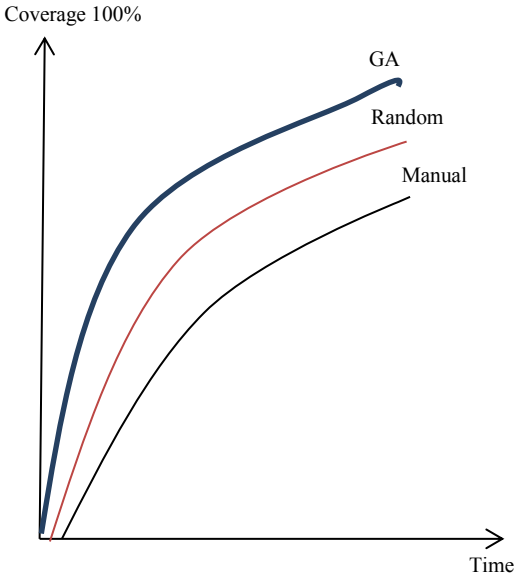


Fig. 5 The GA performance. The methodology collects the coverage data and analyze them, then it assigns a fitness value to each solution.

TABLE III
GA-BASED TEST GENERATION RESULTS TO GET 100% COVERAGE

Method		Random testing		Our GA Approach	
Design	# Scenarios (100 % coverage)	# Stimulus	Run time (s)	# Stimulus	Run time (s)
#1	4	120	3	100	2
#2	16	200	4	150	2.6
#3	6	130	3.2	90	1
#4	12	180	3.5	110	1.3
#5	8	190	3.7	120	1.5
#6	10	195	3.8	124	2.1
#7	6	130	3	120	2.2
#8	18	210	4	155	2.6
#9	8	180	3.7	96	1.6
#10	14	190	3.5	114	1.5
#11	10	170	3.2	111	1.7
#12	12	215	3.2	144	2.4

V. CONCLUSIONS

Bug localization is a process of identifying the specific locations or regions of source code that is buggy and needs to be modified to repair the defect. Bug localization can significantly reduce human effort and design cost. In this paper, a novel automated coverage-based functional bug localization method for complex HDL designs is proposed which significantly reduces debugging time. The proposed bug localization methodology takes information from regression suite as an input and produces a ranked list of suspicious part of code. Our methodology is a promising solution to reduce required time to localize bugs significantly.

Moreover, an online RTL-level scan-chain methodology is proposed to reduce debugging time and effort for emulation. Run-time modifications of the values of any of the internal signals of the DUT during execution can be easily performed through the proposed online scan-chain methodology. A utility tool was developed to help ease this process. Our experiment shows that, the area overhead is neglected compared to the gained performance benefits. But, IP design requires more compilation time.

The main challenge in using constraint random testing (CRT) is that manual analysis for the coverage report is needed to find the untested scenarios and modify the test cases to achieve 100% coverage. We need to replace the manual effort by an automatic method or a tool that will be able to extract the coverage report, identify the untested scenarios, add new constraints, and iterate this process until 100% coverage is attained. In other words, we need an automated technique to automate the feedback from coverage report analysis to test generation process. In this chapter, the implementation of this automatic feedback loop is presented. The automatic feedback loop is based on artificial intelligence technique called genetic algorithm (GA). This technique accelerates coverage-driven functional verification and achieves coverage closure rapidly by covering uncovered scenarios in the coverage report (coverage holes).

REFERENCES

[1] http://www.itrs.net/.
[2] K. Constantinides, O. Mutlu, and T. M. Austin, "Online design bug detection: RTL analysis, flexible mechanisms, and evaluation," in International Symposium on Microarchitecture (MICRO), 2008, pp. 282–293.
[3] S. B. Park and S. Mitra, ''IFRA: Post-silicon bug localization in processors,'' in Proc. HLDVT, 2009, pp. 154–159.
[4] K. Chang, I. Wagner, V. Bertacco, I. Markov "Automatic Error Diagnosis and Correction for RTL Designs" HLVDT, 2007.
[5] S. Mirzaeian, F. Zheng, K. Cheng "RTL Error Diagnosis Using a Word-Level SAT-Solver" ITC, 2008.
[6] W. E. Wong, V. Debroy, and B. Choi, ''A family of code coverage-based heuristics for effective fault localization,'' J. Syst. Software, vol. 83, no. 2, pp. 188–208, 2010.
[7] W.E. Wong, T. Wei, " A Crosstab-based statistical method for effective fault localization" Proceedings of the First International Conference on Software Testing, Verification and Validation (ICST), Lillehammer, Norway, April 2008, pp. 42–51.
[8] J. A. Jones and M. J. Harrold "Empirical evaluation of the Tarantula automatic fault-localization technique", Proc. Int. Conf. on Automated Software Engineering, pp. 273-283, 2005.
[9] J. Rau, C. Chien, and J. Ma, "Reconfigurable Multiple Scan-Chains for Reducing Test Application Time of SOCs", ISCAS 2005.
[10] I. Mavroidis and I. Papaefstathiou, "Accelerating emulation and providing full chip observability and controllability", Design & Test of Computers, IEEE, Dec. 2009.
[11] I. Mavroidis, I. Papaefstathiou, "Efficient Testbench Code Synthesis for a Hardware Emulator System", DATE 2007, Nice, France.
[12] S. Banerjee, T. Gupta "Efficient Online RTL Debugging Methodology for Logic Emulation Systems" 25th International Conference on VLSI Design, 2012.
[13] Yingpan Wu, Lixin Yu, Wei Zhuang and Jianyong Wang, "A Coverage-Driven Constraint Random-Based Functional Verification Method of Pipeline Unit," Computer and Information Science, ACIS International Conference, pp. 1049-1054, 2009.
[14] Benjamin.M, Geist. D, Hartman. A, Wolfsthal.Y and Mas. G,Smeets.R." A study in coverage-driven test generation "Design Automation Conference, 1999. Proceedings. 36th Issue pp. 970 – 975, 1999.
[15] Fine. S and Ziv. A." Coverage Directed Test Generation for Functional Verification using Bayesian Networks" Design Automation Conference, 2003. Proceedings Issue Date: 2-6 June pp. 286 – 291.
[16] Zaer S Abo-Hammour, Othman MK Alsmadi, and Adnan M. Al-Smadi "Frequency-Based Model Order Reduction via Genetic Algorithm Approach" 7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA), 2011.
[17] Z. S. Abo-Hammour, O. M. Alsmadi, and A. M. Al-Smadi "Frequency-Based Model Order Reduction via Genetic Algorithm Approach" 7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA), 2011.
[18] I. Yun, L. A. Carastro, R. Poddar, M. A. Brooke, G. S. May, K-Sook Hyun, and K. E. Pyun "Extraction of Passive Device Model Parameters Using Genetic Algorithms" ETRI Journal, Volume 22, Number 1, March 2000.
[19] K. Thirugnanam, E. Reena, M. Singh, P. Kumar "Mathematical Modeling of Li-Ion Battery Using Genetic Algorithm Approach for V2G Applications" IEEE Transactions On Energy Conversion, VOL. 29, NO. 2, JUNE 2014.