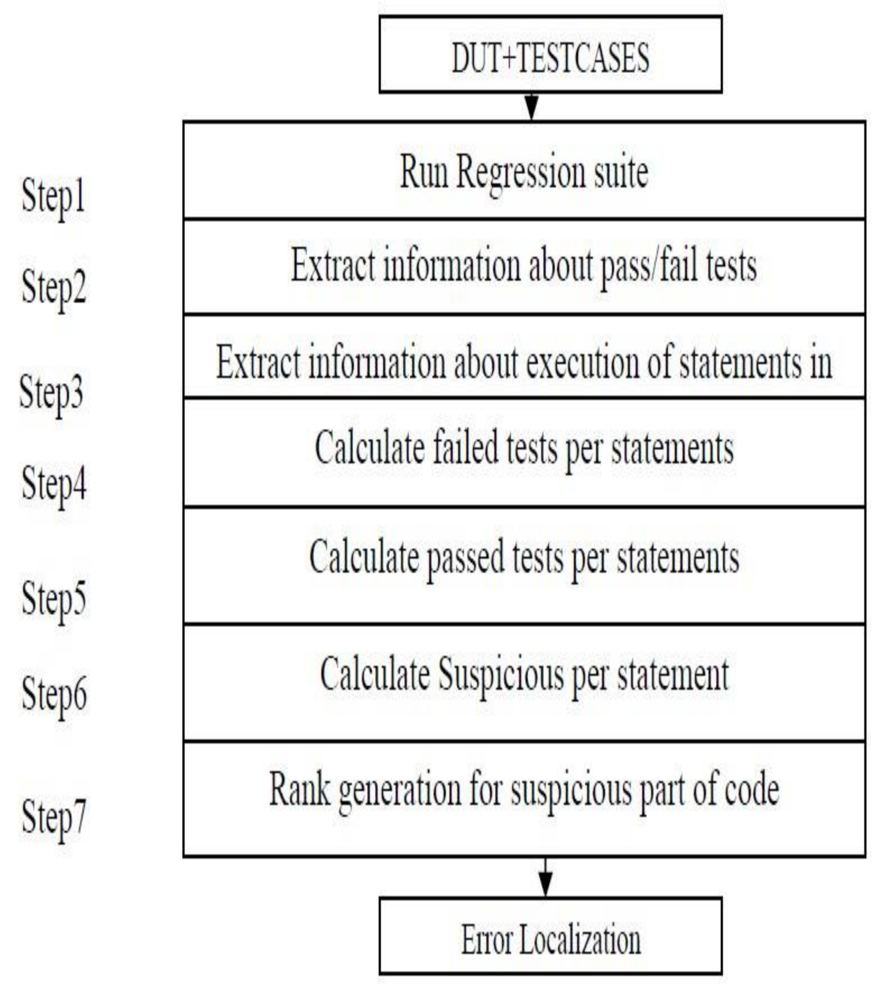
A New Trends in RTL Verification: Bug Localization, Scan-Chain-Based Methodology, GA-Based Test Generation

Khaled Salah
Mentor Graphics

RTL BUG LOCALIZATION



The proposed methodology for automated bug localization

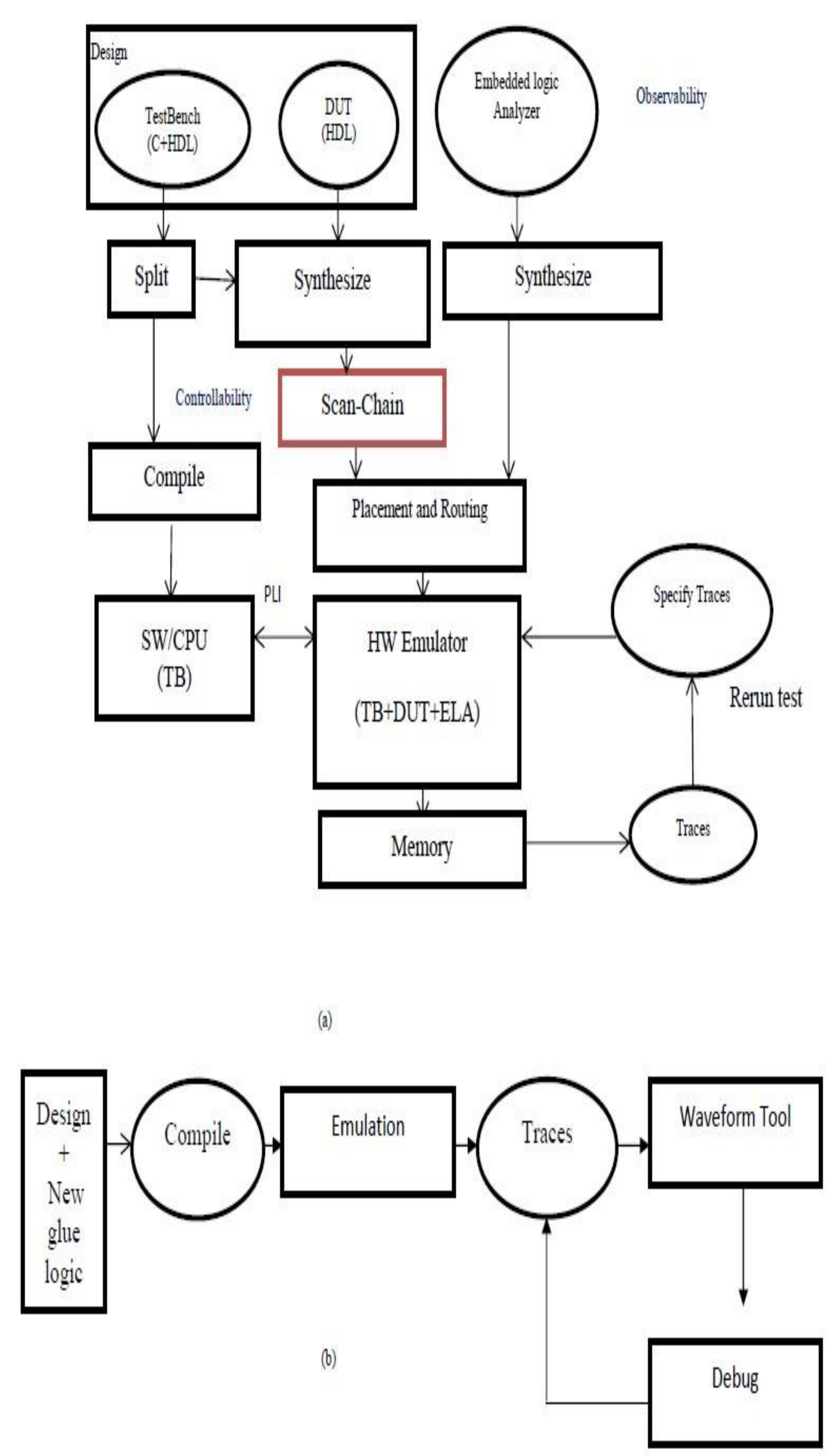
A MOTIVATIONAL EXAMPLE TO DESCRIBE THE PROPOSED METHODOLOGY.

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	

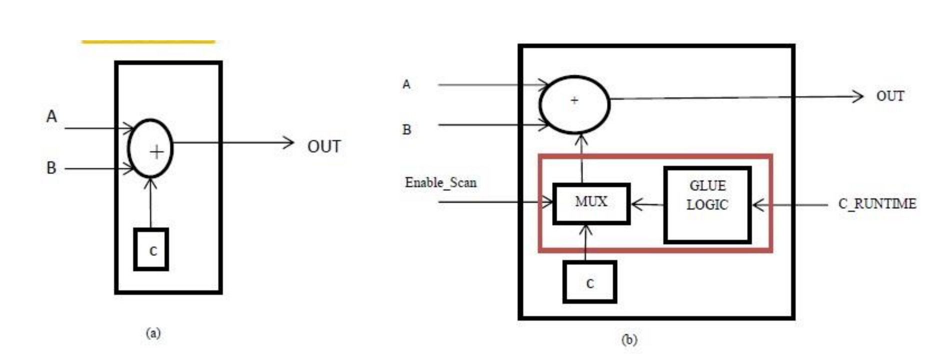
BUG LOCALIZATION TIME USING THE PROPOSED METHODOLOGY VERSUS MANUAL DEBUG IN A COMPLEX DESIGN, WHICH CONTAINS MORE THAN 5000 LINE OF RTL CODE

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;

THE PROPOSED RTL-LEVEL SCAN-CHAIN METHODOLOGY

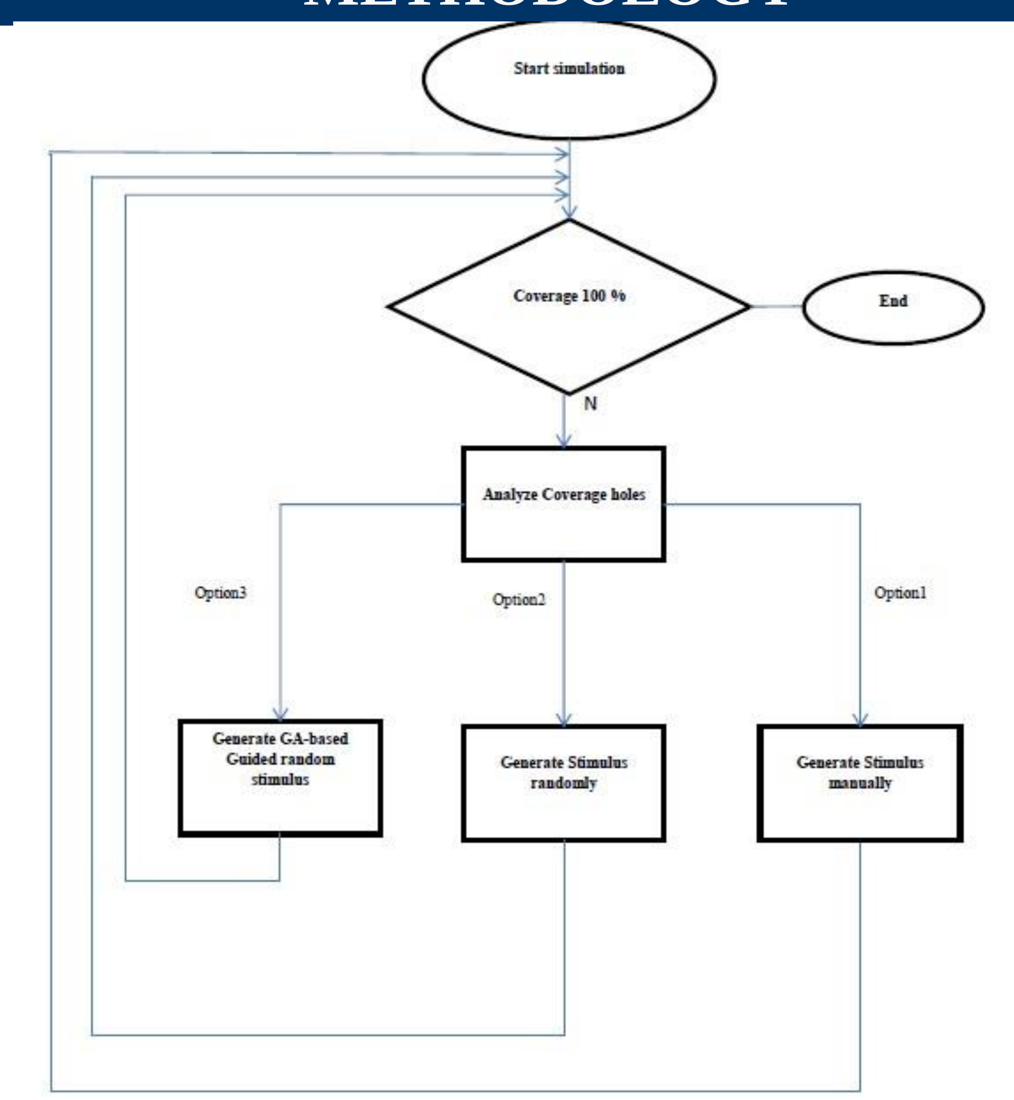


Proposed Emulation Flow (online flow), synthesizable testbench methodology, scan-chain methodology, a) detailed, (b) simplified.

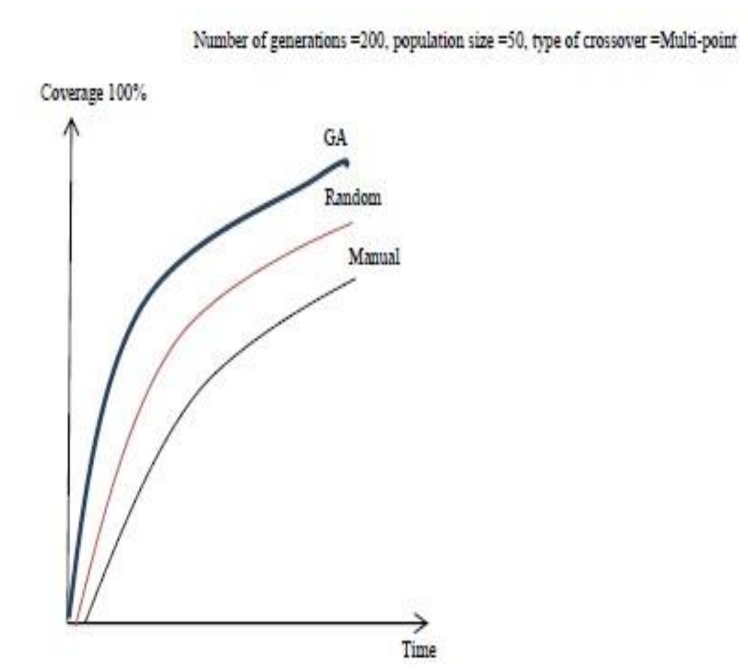


(a) Normal design example, (b) proposed scan-chain methodology for the design example in (a).

GA-BASED PROPOSED METHODOLOGY



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



The GA performance

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

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 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 (coverage holes).