Simulation-based Fault Injection on Ibex Core with UVM Environment

Li Lu[1,2]*, Junchao Chen[1], Markus Ulbricht[1] and Milos Krstic[1,2]

[1]IHP-Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany
[2]University of Potsdam, Potsdam, Germany

Abstract

This paper presents a procedure for implementing simulation-based fault injection on the Ibex core with its UVM testbench. The simulation aims to identify the critical flip-flops where faults could lead to erroneous system operation. We first select testcases for the simulation, based on their contributions to functional coverages. Then the simulation is conducted at the RTL. For each testcase, we remove the identified critical flip-flops from its fault list to reduce the number of faults we need to inject. The identified critical flip-flops are mapped from RTL to the gate level eventually. The procedure could reduce the time required by fault injection to some extent.

Introduction

The probability of system failures in microprocessors, caused by soft errors, increases as the complexity of SoCs increases. Simulation-based fault injection is a commonly used method for circuit reliability analysis in the early design stage. It enables us to identify the vulnerable parts of a circuit and employ fault mitigation techniques to improve the reliability of circuits before manufacturing.

UVM is the most popular simulation verification methodology used in the industry today. Testbenches based on UVM support coverage-driven verification can balance verification completeness with minimum verification effort and time. Ibex [**Schiavone:2017**] is an open-source 32-bit RISC-V CPU core designed for embedded control applications. It is verified using a UVM-based testbench [**ETH Zurich:2018**], which uses the open-source RISCV-DV random instruction generator to generate instructions. The testbench compares the trace log of the Ibex core to the trace log generated by a golden model ISS to check whether the program is executed correctly.

This work presents a procedure to conduct simulation-based fault injection at the gate level on the Ibex core with the UVM testbench to identify critical flip-flops which determine the core's correct functioning. The workflow could reduce the time involved in the fault simulation from three aspects. 1) We first implement the simulation at the RTL to identify critical flip-flops and then map these flip-flops to the gate level (GL). 2) We select testcases from the testcases provided by the UVM testbench based on their contributions to the functional coverage to reduce the number of testcases required by the fault injection. 3) We use fault pruning to reduce the faults we inject for each testcase.

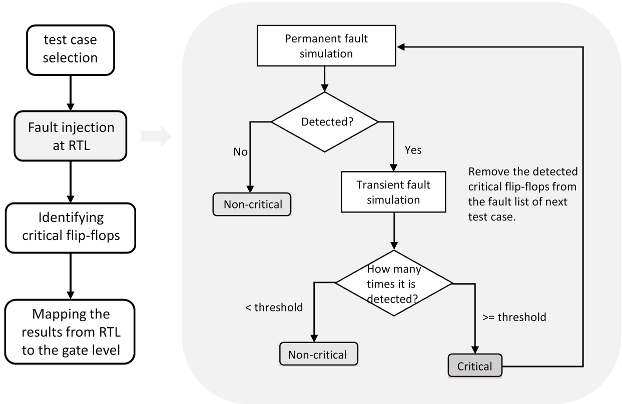
test case selection
Fault injection at RTL
Identifying critical flip-flops
Mapping the results from RTL to the gate level
Permanent fault simulation
Detected?
No
Yes
Non-critical
Transient fault simulation
Remove the detected critical flip-flops from the fault list of next test case.
How many times it is detected?
< threshold
>= threshold
Non-critical
Critical

Figure 1: *The process of simulation-based fault injection.*

Methodologies

The process of simulation-based fault injection is shown in Figure 1. We use Integrated Metrics Center(IMC) provided by Cadence to analyze the contribution of each test case provided by the UVM testbench to the functional coverage. We remove the testcases which don't or less contribute to the coverage and merge some of the testcases to reduce the number of testcases from 39 to 6. The selected testcases are listed in Table 1 and cover around 98.2% functions. Correspondingly, the fault simulation is implemented at the RTL first to identify the critical flip-flops in the Ibex core. These flip-flops are mapped from RTL to the gate level eventually.

We use the Xcelium fault simulator provided by Cadence for the fault simulation. This simulator enables us to reuse the UVM functional verification testbench to build the fault simulation testbench. In our experiments, the simulator compares the value of primary output signals in Ibex when it is fault-free with the value when a fault is injected. The simulator annotates the fault as detected if there is any difference.

*Corresponding author: `lu@ihp-microelectronics.com`

Table 1: *Fault simulation results on Ibex at the RTL.*

Test Name	Simulation Time (ns)	Functional Coverage	Number of Flip-flops Injected Permanent Faults	Number of Flip-flops Injected SEU Faults	Number of Detected Critical Flip-flops	Number of Runs	Time (hours)
rand_jump	59,400,814	4598	2227	943	590	23314	5196
pmp_basic	3,611,144	177	1637	954	62	8044	81
debug_branch_jump	21,970,704	10	1575	349	53	6640	338
mem_error	9,818,104	8	1522	442	327	5942	206
debug_ebreakmu	13,274,244	3	1195	179	101	2627	80
invalid_csr	2,261,544	2	1094	14	0	2258	23

Otherwise, the fault will be annotated as undetected.

RTL vs. GL

Xcelium fault simulator supports fault injection at the RTL and the gate level (GL). We investigate the difference between the fault simulation results at the RTL and GL to prove the process in Figure 1 is feasible. There are 2110 flip-flops matched at the RTL and the GL, 117 flip-flops existing only at the RTL, and 16 at the GL. For flip-flops existing only at the RTL, they are removed by the optimized function of the synthesis tool. Flip-flops only existing at GL are these defined as an enumeration type at RTL, which is not supported by the simulator. Table 2 compares the results and time of fault simulation at the RTL and the GL. The difference between them is minimal for all fault types. For example, only 3.3% (7/2110) flip-flops have different results on Stuck-at-0. The time (based on a single-core Intel Xeon processor E5-4627V2) required for fault simulation at the RTL is less than at the GL, especially for SEU.

Table 2: *Comparision of fault simulation at the RTL and the GL.*

Fault type	Difference	Number of runs		Time (hours)	
		RTL	GL	RTL	GL
Stuck-at-0	7 (≈ 3.3%)	2227	2126	28	34
Stuck-at-1	13 (≈ 6.2%)	2227	2126	26	31
SEU	10 (≈ 4.7%)	18860	19080	1685	2493

Fault Injection

Fault simulation is conducted in the order of the testcase's contribution to the functional coverage from high to low (see Table 1). For each testcase, as shown in Figure 1, we first inject permanent faults (i.e., stuck-at-1 or stuck-at-0) into all flip-flops. If the fault is not detected on the outputs, we can interpret that the fault is masked or the flip-flop is not covered by the testcase. The flip-flop is therefore considered non-critical. Otherwise, transient faults are injected several times in a specified time window. We only inject SEUs here for transient faults because SETs could be masked by flip-flops that are clock-event sensitive. The flip-flop will be identified as critical if the number of detected SEU faults is larger than the threshold we define.

If the flip-flop is annotated as critical, it will be removed from the fault list of the following testcase. In Table 1, the 1st testcase identifies 590 critical flip-flops. And these flip-flops are removed from the fault list of the 2nd testcase. The number of flip-flops for permanent fault injection is reduced from 2227 to 1637, and similar procedures are conducted on other testcases. Table 1 also presents the simulation time of each testcase, the time required by fault simulation, the number of identified critical flip-flops, etc.

Discussion

The procedure could reduce the time involved in the fault simulation as described to some extent. There are some limitations to this approach. We need to specify the threshold to identify critical flip-flops correctly at the beginning. Otherwise, the simulation from the following testcase might be based on an incorrect fault list. The testcases should also be selected based on the relevant application cases they cover. For example, testcase4 randomly inserts instruction fetch or memory load/store errors. If the function is not the concern, it should also be removed from the testlist. The simulation time of testcases could be reduced if we could use more efficient constraints. The efficiency could be improved further with these considerations.

References

[1] Schiavone, Pasquale Davide, et al. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications.", 27th IEEE PATMOS.

[2] ETH Zurich and University of Bologna, 2018, Available: https://ibex-core.readthedocs.io/en/latest/03_reference/verification.html