Assisting Fault injection simulations for Functional Safety signoff using Formal

Pulicharla Ravindrareddy, Analog Devices, Bengaluru, India
(*pulicharla.ravindrareddy@analog.com*)

Abstract— With formal verification being part of routine design verification practices, it is essential to bring out more value from formal verification beyond exercising assertions and bug hunting. One such area is Functional Safety Verification. This paper discusses the Formal Techniques that can be used in assisting the Fault Injection simulations for Functional safety signoff.

I. Introduction

Functional safety is a growing concern for automotive industry and the components that semiconductor vendors providing are involved in crucial decision-making process of saving lives. ISO 26262:2011 is the functional safety standard for the automotive industry. The standard imposes Fault Injection as a predominant method in ensuring the hardware, software and their integration interface is functionally safe. Designs must have required built-in system-level safety mechanisms to catch all the faults that could occur due to various reasons. Different Formal analysis techniques can be used to improve the productivity of the fault injection campaign.

II. Formal Techniques

Two Formal techniques can be used to improve the productivity of the fault injection campaign, one, before the start of the fault injection campaign and the other, after the fault injection campaign.

1. Formal Fault Testability Analysis (before the start of the fault injection campaign)
2. Formal Fault Detection Analysis (after the fault injection campaign)

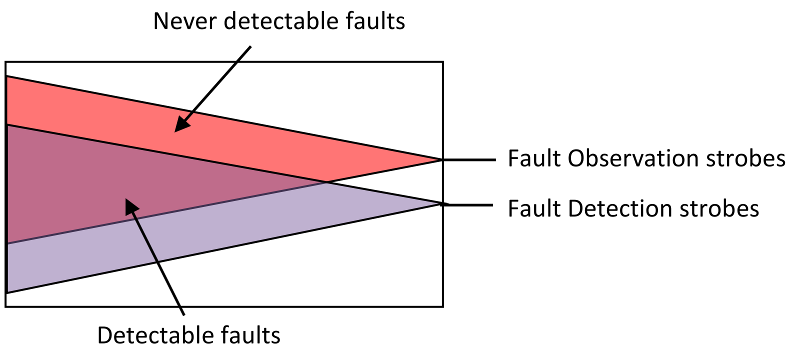
A. *Formal Fault Testability Analysis:*

With increasing design size and complexity, the number of faults to be injected and campaigned are sky rocketing. Many useless fault simulations are being run in the campaign which do not have any impact on the safety critical areas of the design. So, Formal can be used to remove these useless fault simulations. It helps in finding out the un-testable and un-activatable faults (safe faults) whose effect cannot be observed at design safety critical areas independent of the stimulus.

- Un-testable fault nodes: Fault nodes which do not have physical connection to the safety critical elements.
- Un-activatable fault nodes: An SA0 or SA1 fault injected on a node that is a constant 0 or 1 respectively, do not have any functionality change.

It uses the concept of Cone of Influence (COI) in determining the un-testable faults (faults that are out of COI of observation strobes) and Constant analysis to determine the un-activatable faults based on the provided constraints set. These untestable and un-activatable faults are functionally safe and can be removed from the fault injection campaign.

COI analysis can also be helpful in further reducing the number of faults that are considered for fault campaign, by finding out the never detectable faults. The overlapped COI of fault observation strobes and detection strobes gives an understanding of the quality of the design and the safety mechanisms existing at the system level.



The faults under “Never detectable faults” cannot be detected by the Fault Detection strobes. As these faults are never detectable, there is no meaning of running these fault simulations. So, these faults also can be dropped, and the team need to review these faults to create more overlap between Observation strobes and Detection strobes.

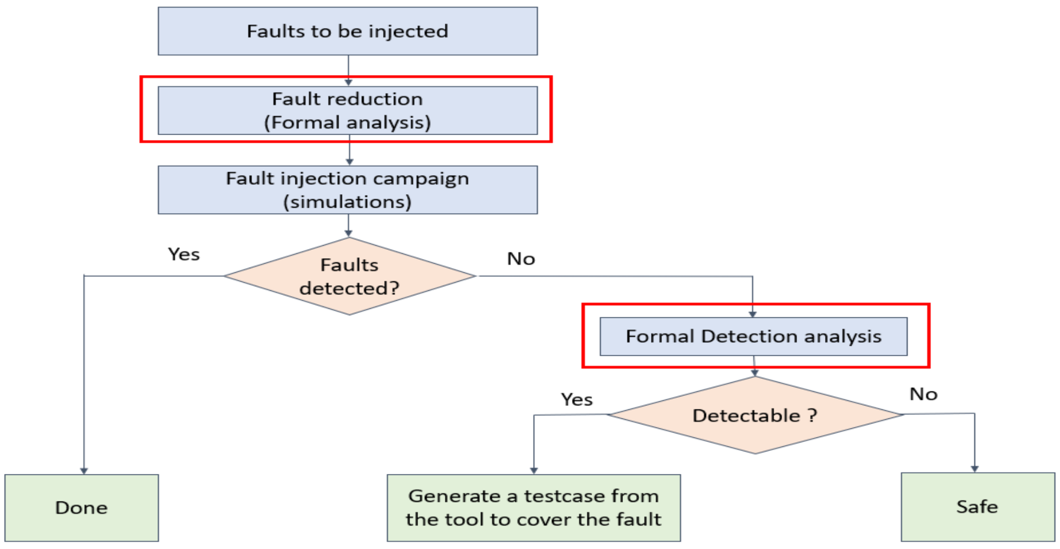
Sometimes, given that the block owner’s limited knowledge of the IP block/design, it is possible to miss a strobe or add an un-wanted strobe. Formal reports based on COI would also let us confirm the provided strobes for the design are precise.

B. Formal Fault Detection Analysis:

Once we get a meaningful subset of faults, fault campaign is run using a fault injection simulator. Fault injection simulator covers the injected faults based on the applied stimuli. Functional simulation regression environment consists of scenarios that test functionality of the design, while a fault detection analysis requires scenarios that exercises the design for fault’s propagation and detection. Assuming a fault is not detected after the campaign it is tedious for someone to think through a scenario that would exercise the fault so that it is propagated and detected.

Formal on the other hand tries to cover the faults with all possible stimuli. It is good at introducing the faults and to know the scenarios which can propagate and detect the faults. Formal detection analysis can be used once the fault injection simulations are done to know if any undetected fault can be detected or never detectable. This is very handy in debugging why a fault is not propagated or not detected after a safety campaign. Thus, we recommend using Formal techniques to improve the process and productivity during debug.

III. FLOW DIAGRAM


Faults to be injected
Fault reduction (Formal analysis)
Fault injection campaign (simulations)
Faults detected?
Yes
No
Formal Detection analysis
Detectable ?
Yes
No
Done
Generate a testcase from the tool to cover the fault
Safe

IV. Effort required for applying Formal

1. Setup: Formal verification is being used in almost all the designs for some or the other verification tasks. So, Formal setup would have been already available at both block and system level. In our case, setup was already available at both block and system level. We used the same setup and added the faults to the Formal tool that are targeted for fault injection. It is not of much effort needed here.
2. Runtimes:
    a. Formal Fault Testability analysis: This is a structural analysis of the design with respect to the strobe points. Structural analysis does not take much time. In our case, it hardly took 2min to give out the results.

    b. Formal Fault Detection analysis: This is a functional analysis. In this case, Formal tool has to run both good and fault simulations. The tool compares the results of good and fault simulations at all strobe points until a change in the strobe points is detected. The analysis is done at block level and the runtimes are considerably more in our case, approximately 2 hours for each fault. Later learned that our design follows a serial protocol and it is expected that the tool can take more time. If the design follows a parallel protocol and a control heavy oriented design, it is not expected to take more runtimes at block level.

V. Challenges faced

We did not face much problems for the Formal Testability analysis, because we already had Formal setup up and running. But we did face few challenges for Formal detection analysis. Once the Formal detection analysis proves that the fault can be detectable, there are two challenges.

1. Designer needs to analyze the waveform and confirm that the tool exercised a legal scenario
2. DV engineer needs to write a testcase for that scenario

For creating testcase, first we tried saving the waveform in .vcd format. We have a script to convert .vcd to a Verilog testcase. We generated the testcase and later found that .vcd does not have port information of the signals. So, we have dropped this idea and created a directed testcase which took us some effort. This directed testcase also covered some other faults which can be covered from the same scenario. We are exploring more on the creation of testcase once the formal says that the fault can be detected.

VI. Results

	Total number of faults	**Safe faults**	**Never detectable faults**
Fault Injection simulations	57234	9.19%	30.7%

From the fault injection simulations, around 20% of the faults are classified as undetected. These undetected faults are run using Formal Detection analysis to check if the faults can be detected or never detected. Generated a testcase for the faults that are detected by Formal detection analysis and used the testcases for running fault simulations, which reduced the number of undetected faults to 12%.

VII. Conclusion

The aim of this paper is to utilize formal verification in such a way that it will enhance simulation productivity beyond its primary goal of verifying design functionality.