Are My Fault Campaigns Providing Accurate Results for ISO 26262 Certification?

Hyunsun Ahn, Samsung Electronics Co., Ltd., Korea (hyunsun.ahn@samsung.com)
James Kim, Siemens EDA, Korea (james.kim@siemens.com)
Arun Gogineni, Siemens EDA, USA (arun.gogineni@siemens.com)
Ann Keffer, Siemens EDA, USA (ann.keffer@siemens.com)
MyungKyoon Yim, Samsung Electronics Co., Ltd., Korea (mk.yim@samsung.com)
Soobon Kim, Samsung Electronics Co., Ltd., Korea (soobon.kim@samsung.com)
Youngsik Kim, Samsung Electronics Co., Ltd., Korea (ys31.kim@samsung.com)
Seonil Brian Choi, Samsung Electronics Co., Ltd., Korea (seonilb.choi@samsung.com)

***Abstract*- Running an efficient fault campaign with accurate results can be a difficult task, especially on large complex designs, but optimizations and techniques exist to make it less so. Satisfactory results may have been attainable in the past, but the increased size and complexity of automotive designs, the difficulty in providing good-quality stimulus, and the large number of faults that need to be analyzed make it impractical to perform exhaustive, safety verification using a single technology. Fortunately, there are many methodologies that provide efficiencies for running fault campaigns starting with early and accurate safety analysis. Using a safety analysis tool that structurally analyzes RTL and provides accurate safety metrics, such as FIT and DC, allows engineers to understand the safety level of their design and reach the optimal safety architecture prior to costly fault injection. This paper will discuss these techniques using Samsung's safety flow methodology as a real-world example. It will outline real use cases, discuss the importance of accurate metrics during the analysis phase of the flow, outline fault list optimization techniques, propose recommended set up conditions for optimal stimuli, and discuss when good-machine simulation is needed based on Samsung's certification activity.**

I. Introduction

In the paper "Complex Safety Mechanisms Require Interoperability and Automation for Validation and Metric Closure" presented at DVCon US 2023, we covered a fundamental methodology to overcome typical fault campaign challenges for SoCs with complex safety mechanisms. Whereas the paper described a general methodology and flow for fault campaigns, it did not go into the details of how to plan and verify fault campaign activity.

Questions not addressed in that paper include:

- How do we know the fault campaigns we are doing are providing accurate results?
- Are we running fault campaigns that represent real use cases?
- Is the stimulus behavior at the RTL providing results that match the final gate-level netlist results?
- What are the different techniques available to accelerate fault campaigns?
- Do we need to do good-machine simulation for all the stimulus at all levels of SoC development?

In this paper we attempt to answer these questions and identify what is required in fault campaign methodologies to achieve accurate results for ISO 26262 certification. We further explain what should be considered to make sure good-quality stimulus is generated for fault campaigns, how to create stimulus quality checks prior to running fault campaigns, and how to implement a methodology for good-machine simulation. Fault simulation is resource intensive. At a minimum, you need to simulate a good machine and a faulty machine. The good machine is used as a reference to determine if any of the faults are detected. Good-machine simulation is where all state elements and observation points are compared at all times in the original stimulus to catch dormant deviation.

Good-machine simulation is a systematic way to ensure the stimulus is good quality and provides accurate results. Good-machine simulation can solve the missing simulation data problem and reduce the load on the functional verification team. To create stimulus that provides accurate results at both the RTL and gate level, engineers need to make sure that the stimulus has minimal to no race conditions and no behavioral mismatches between simulation and synthesis. Making sure good stimulus is used for fault campaigns and having the right type of set up conditions to generate the stimulus to test real use cases is critical. An integrated set of tools such as simulation and emulation for fault injection also accelerates fault campaign runs as we will illustrate with the Samsung design.

Additionally, although gate level simulation is more accurate and more closely represents silicon behavior, using RTL models is an acceptable approach provided that the RTL code used is shown to be well correlated with the gate level netlist. Kaleidoscope, Siemens EDA’s concurrent fault simulator, utilizes a pseudo synthesis method for RTL simulation. This means that in essence we are running a gate level fault campaign.

Another important aspect of an efficient fault campaign is making sure the fault lists are optimized. We automatically create optimized, customized fault lists for the appropriate engine, and run concurrent, parallel fault injection runs that use all available compute power to reduce the cost of fault injection.

Fault optimization techniques used on the Samsung design include:

- Safety mechanism aware
- Fault collapsing
- Statistical random sampling

Safety-mechanism-aware optimization eliminates faults that are not part of the safety mechanism's cone of influence (COI). Fault collapsing eliminates logically equivalent faults, and statistical random sampling allows you to reduce the fault list size with a random sampling of faults estimated given a confidence level. All these methodologies for optimizing fault lists were considered.

II. Good-Quality Stimulus

There are several steps to making sure good-quality stimulus is used to get optimal results from a fault campaign. To generate proper stimulus, several conditions must be considered: correct design setup, forces in the testbench, RTL simulation versus synthesis mismatches, Xprop(resolve), and race handling.

Correct Design Setup

When using the stimulus generated for good-machine checks and fault campaigns, we need to make sure we have a correct setup. The design should be the same between functional stimulus generation and good-machine/fault campaign simulation.

For quality-stimulus we need to make sure to use the same input configuration, including:

- File lists
- Defines
- Parameters
- Preloads into memory

Forces in the Testbench

A complete list of forces that were used in functional simulation needs to be passed on to the fault campaign engine. A force on a signal net should propagate to wherever the net goes across all hierarchies, as shown in below example:

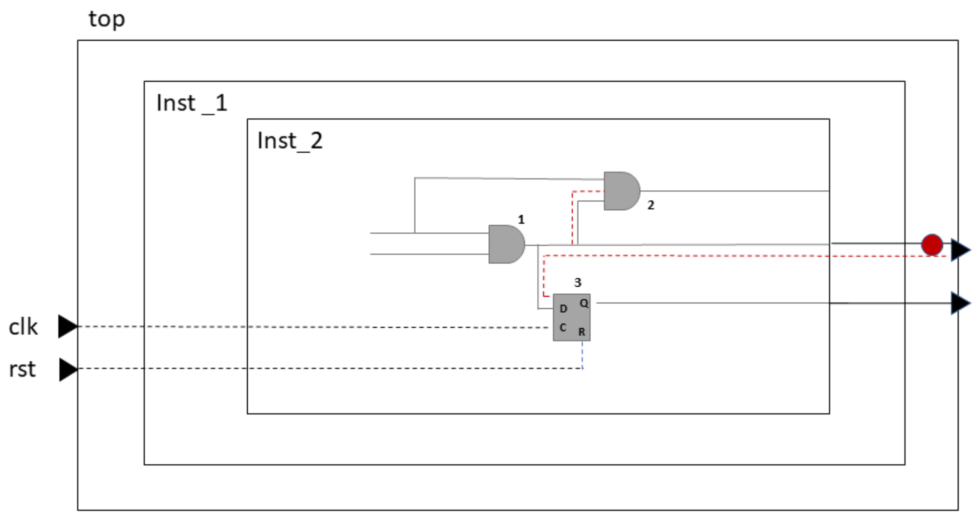
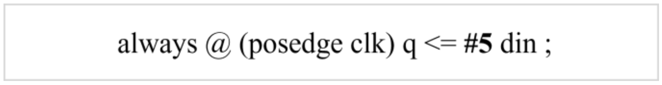


Figure1. Valid force value propagation through design for fault campaign

In the Figure 1, the force on the net is shown by the red circle. The effect of this force should be propagated to the output of gate 1, which will affect gate 2 and flip flop 3. The force value propagation cannot stop at top.inst_1 or top.inst_1.inst_2. This forcing behavior can be different between RTL and Gate Level Netlist. Synthesized netlist behavior is closer to silicon because synthesized design has flattened nets similar with silicon.

RTL # Delays

When doing functional simulation, it is always a good practice to avoid putting # delays in the RTL, as shown in figure 2, because adding delays will result in simulation versus synthesis mismatches in the functionality of the design.



```
always @ (posedge clk) q <= #5 din ;
```

Figure 2. # Delays in RTL

To make sure to avoid any inadvertent # delays, stimulus generation should always use non # delay modeling. All the industry standard simulators have configurations for this. In case the original code already has lots of # delays, additional handling is required to guarantee valid fault campaign results.

Simulation Behavior versus Synthesis Mismatches

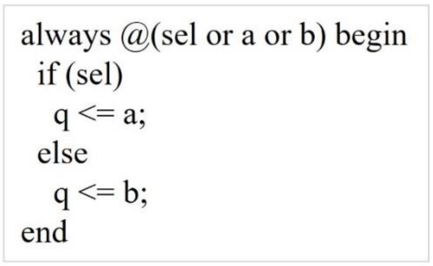
Simulation model mismatches occur due to discrepancies between the simulation model and the actual hardware behavior. These discrepancies can cause unexpected results, which can occur due to incorrect logic or improper design implementation at the RTL. Some examples of the effects of this behavior are:

- Missing synchronizers between clock domains
- Improper simulation models that do not match the final hardware implementation
- Out of indexing of data arrays
- Mismatches between address pointers and data location size

To avoid these situations, we recommend using lint-clean RTL for stimulus generation. Any industry standard linter can be used to achieve good results.

XPROP=Resolve in RTL Stimulus Generation

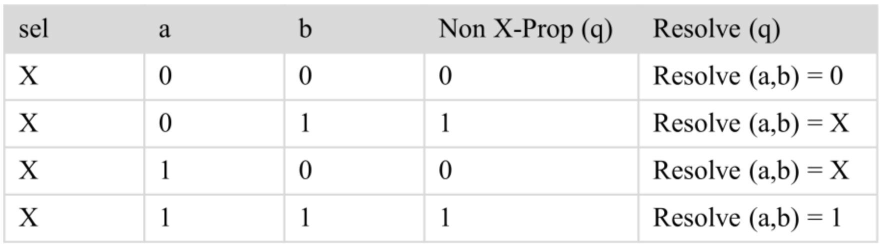
RTL simulation uses the value of X to temporarily represent a state that cannot be predicted or determined at a given time. Because X values do not appear in the design once it has been implemented in silicon, a mismatch can occur between simulation and silicon. Typically, RTL simulation is done based on X-Optimism, but to reflect silicon behavior more closely, X-Pessimism is preferred when debugging X behavior in simulation. This also be one of the best practice can show that used RTL correlated with gate level netlist when using RTL for fault injection testing. Consider the following code in figure 3:



```
always @(sel or a or b) begin
  if (sel)
    q <= a;
  else
    q <= b;
end
```

Figure 3. X-optimism, X-pessimism code example

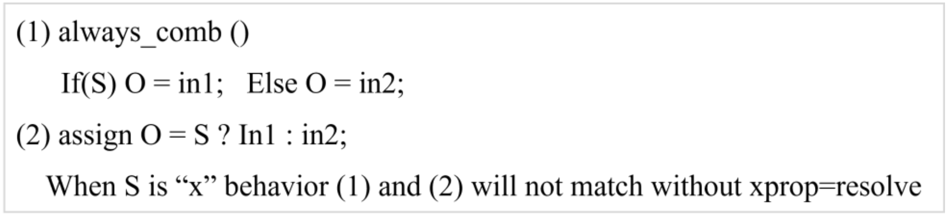
Table 1. shows simulation results for both X-Optimism and X-Pessimism. Resolve X is the behavior of X-Pessimism.



sel	a	b	Non X-Prop (q)	Resolve (q)
X	0	0	0	Resolve (a,b) = 0
X	0	1	1	Resolve (a,b) = X
X	1	0	0	Resolve (a,b) = X
X	1	1	1	Resolve (a,b) = 1

Table 1. Simulation result by mode of X-propagation (Non-X propagation versus X resolve mode

Use XPROP=resolve in RTL simulation to attempt to resolve X values by forcing them to deterministic states based on certain algorithms or resolution rules. This helps in modeling how the hardware might behave when encountering unknown states. It also provides better insight into how the synthesized hardware would behave when encountering unknown states and aids in catching issues that might arise due to gate-level optimizations or mismatches between RTL and gate-level representations as shown in figure 4.



```
(1) always_comb ()
      If(S) O = in1;   Else O = in2;
(2) assign O = S ? In1 : in2;
    When S is "x" behavior (1) and (2) will not match without xprop=resolve
```

Figure 4. X-propagation simulation result on MUX design

Race Conditions in Stimulus

Races can cause deviations in simulation behavior. The most common races are.

- Reset to clock race

This race condition happens when the testbench changes the reset and clock at the same time. This can be avoided by changing the reset when the clock is active. We have observed that this is a fairly hard thing to do in complex testbenches. We added a feature to the Siemens EDA's KaleidoScope tool to delay the reset until the active clock edge is absorbed.

- Data to clock race

This happens when the testbench drives data on an active edge of the clocks. To avoid this, data should always be driven on an inactive edge of the clock. If a user doesn't have a clear way of handling this, to take advantage of the features of used simulator can be used when generating stimulus. For example, QuestaSim Siemens's digital simulator provides **-deraceclockdata** switch for simulation, it can mitigate race between clock and data and provides a way to read old (stable) data values.

III. Good-Machine Simulation

A fault campaign tool in its good-machine simulation must check to see that all the state elements of the design are matching with that of the input stimulus. This includes checking observe points and all state elements for deviations.

Checking Observe Points

Limiting the check to only observe points will compromise the correctness of the fault campaign results because dormant deviations of state elements can result in incorrect behavior.

Criticality Checking All State Elements

The KaleidoScope fault simulator checks not only observe points but also all state elements for deviations. The benefit of this methodology is that it avoids dormant deviations. Since KaleidoScope is checking all state elements and observe points for deviations, the results of fault campaigns are always accurate.

Kaleidoscope's approach to good-machine simulation consists of the following:

- Check deviations at observe points
- Check deviations at all state elements (flip-flops and latches)
- Check deviations of memory content
- Catch all dormant deviations
- Fault campaign based on correct golden simulation

IV. Fault Campaign

We propose a fault campaign methodology that addresses the correctness and rightness of the campaign. Following this methodology, shown in figure 5, guarantees that the fault campaign is right the first-time.

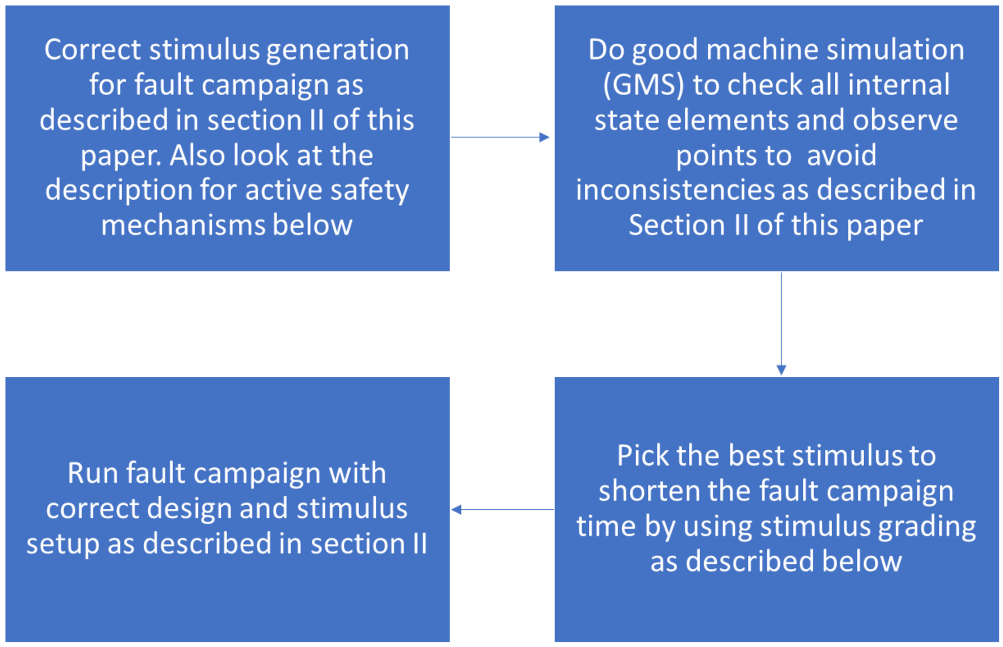

Correct stimulus generation for fault campaign as described in section II of this paper. Also look at the description for active safety mechanisms below
Do good machine simulation (GMS) to check all internal state elements and observe points to avoid inconsistencies as described in Section II of this paper
Pick the best stimulus to shorten the fault campaign time by using stimulus grading as described below
Run fault campaign with correct design and stimulus setup as described in section II

Figure 5. Fault campaign methodology

Active Safety Mechanism

Doing fault campaigns on stimulus where safety mechanisms are not active is detrimental as it will result in marking all the faults as undetected. It is very important to activate all safety mechanisms in the design for fault campaigns.

A good approach is to put self-checking in place when implementing the safety mechanism which will copy the behavior of the fault and will trigger an alarm on the safety mechanism [9]. Use these steps:

- Add an error inject capability in the safety mechanism
- Use this capability to see the proper alarm trigger
- Clear the error injection and clear the alarm (the safety mechanism is now ready to use)
- Check the safety mechanism activation with classified detected fault results

Stimulus Grading

Stimulus grading is a technique where all input stimuli are analyzed to pick the one that allows performing the most fault injections. This can be as simple as looking at toggle coverage, or it may involve propagating faults to the next state element to see if the deviation propagates, which is more complex than looking at toggle coverage.

Race Conditions in Stimulus

We talked about how important it is to make sure to handle race conditions in generating stimulus for fault campaigns. In the same way, it is very critical to also make sure to handle race conditions in the fault injection flow so the results from the fault campaign will be consistent.

Xprop=Resolve

As discussed, we strongly recommended generating stimulus using the resolve x-propagation method.

V. TEST DATA ANALYSIS

Good-Machine Simulation Deviations

In the Samsung design, the initial good-machine simulation analysis showed quite a lot of deviations. After taking care of stimulus generation issues, these deviations went away. The initial stimulus was generated with XPROP=NOXPROP, which resulted in 15% of total state elements deviated. We generated stimulus with XPROP=RESOLVE and used it to do good-machine simulation. The deviation percentage went down to 4.63%. We did not stop here as we wanted to understand all the deviations to see if we could fix/address/understand them. In the next section we will discuss our approach to this.

Stimulus Generate Method	Deviation %
Original stimulus Generated with XPROP=noXprop	~ 15%
Updated Stimulus Generated using XPROP=resolve	4.63%

Table 2. Xprop mode experiment results in good-machine simulation

Resolving Remaining Deviation

The remaining 4.6% deviations were analyzed in several categories. The table below shows several of these categories:

	Cause	**Number of Deviations specific to the cause**
	Total Remaining (4.6%)	37,546
1	Force signal list Missing simulation force list	600 170
2	Race Delay	27,205
3	RTL # Delays	7,722
4	Simulation behavior vs synthesis mismatch	1,845
5	Design should be the same between stimulus generation and good-machine simulation	4

Table 3. Results for resolving good-machine simulation deviations

• The first set of deviations (600+170) are caused by incorrect force list and incorrect force behavior.

• The second set of deviations (27,205) are because of the reset to clock race in the input stimulus. We added support in KaleidoScope to delay the reset; this fixed all the deviations.

• The third set of deviations (7,722) are because of # delays in RTL. We generated stimulus using zero delay simulation.

• The fourth set of deviations (1,845) are because of the out of index in FIFO models

• The fifth set of deviations (4) are because of parameter mismatch between stimulus generation and good-machine simulation. Design set up is updated to fix parameter issue, which fixed these deviations.

Fault Campaign Results

The fault space for the Samsung design in this exercise is huge. Due to the size of the fault space, we generated a subset of faults using the statistical random sampling technique supported by KaleidoScope.

The total number of faults analysed using this technique was 4598 faults. In the next section, we will walk through the results for this fault list.

	Categories	**% Of faults**
1	Alarm Detected	65.87 %
2	Residual	9.86 %
3	No Deviation	14.98 %
4	Not Injected	9.38 %

Table 4. Results of fault campaign

We used the fully validated stimulus from good-machine simulation, which guaranteed the correctness of the results. We injected a total of 100% of faults, and the results were observed in the four categories shown in Table 4. The first category, Alarm Detected, means the 65.87 % of faults were detected by a safety mechanism. The second category, Residual, means the 9.86 % of faults were not detected by a safety mechanism and are considered dangerous faults. But this is an expected result since most of the residual faults were injected on an unprotected area due to a partially configured safety mechanism on the boundary of the target design. The third category, No Deviation, means that 14.98 % of faults were injected but did not cause any deviations in the design for this stimulus. The fourth category, Not Injected, means that 9.38 % of faults were not injected because input stimulus did not allow it. The golden values for those fault nodes matched with that of the fault value throughout the simulation.

VI. Conclusion

Here are the latest experiment results after applying all methods discussed in this paper. The results in Table 5 came from an experiment that was explained in a previous paper[4]. Table 6 shows the updated results after applying all the methods described in this paper including increasing stimulus quality, valid set up, and cleanup of all good-machine simulations. The total fault space used for this experiment is represented as 100% in the Total Faults% column. All other test results are represented as the detected fault ratio of the total fault space.

BUS Logic	Total Faults% (SA0+SA1)	Test 1: Detected (%)	Test 2: Detected (%)	Test 3: Detected (%)	Test 4: Detected (%)	Test 5: Detected (%)
NOC in Safety Island	100%	3.27%	5.50%	9.80%	42.81%	TBD (Future work)

Table 5. Previous results of fault campaign [4]

BUS Logic	Total Faults% (SA0+SA1)	Test 1: Detected (%)	Test 2: Detected (%)	Test 3: Detected (%)	Test 4: Detected (%)	Test 5: Detected (%)
NOC in Safety Island	100%	3.27%	5.50%	9.80%	42.81%	**65.87%**

Table 6. Updated results of fault campaign

In answering the question, “are my fault campaigns providing accurate results for ISO certification?”, we described critical concepts and processes that need to be used for quality fault campaign results. We also addressed the criticality of handling races, forces, delays, design set up concerns, and simulation versus synthesis issues. The analysis shown in this paper is based on the Samsung design where we illustrated how following the methodologies described in this paper will help ensure your fault campaign results are accurate.

REFERENCES

[1] ISO 26262 Part 5: Product development at the hardware level, Second edition 2018-12

[2] ISO 26262 Part 11: Guidelines on application of ISO26262 to semiconductors, Second edition 2018-12

[3] Validating the complex safety mechanisms, Siemens Verification Academy

[4] Complex Safety Mechanisms Need Interoperability for Validation and Close Loop for Final Metrics, Protecting Safety and Security, DVCon US 2023

[5] Austemper KaleidoScope User Guide – Safety Verification

[6] Austemper SafetyScope User Guide – Safety Analysis

[7] Parallel fault simulator with back propagation enhancement, United States patent US 11,036,604 B2. Jun 2021

[8] Fault campaign in mixed signal environment, United States patent US 10,775,430 B2. Sep 2020

[9] Functional Safety Synthesis, United States patent US 10,796,047 B2. Oct 2020