

Virtualized Fault Injection Methods in the Context of the ISO 26262 Standard

Victor Reyes
Synopsys Inc.

ABSTRACT

Software quality is one of the biggest concerns of the automotive industry. Releasing a product with defects and having a recall can have enormous direct and indirect cost for an automotive OEM. In order to improve software quality is not sufficient to only increase the number of tests. It is extremely important to establish more sophisticated tests that can cover corner cases which are not unveiled during normal operation. Typically, corner cases are very difficult to test as those are often only triggered when the underlying hardware fails or the software gets unexpectedly corrupted. How to test those cases, to make sure that the right SW routines are executed and that the system moves back on time to a safe state? Fault-injection methods are typically used to cover a subset of these tests. However, there are quite some limitations on how effective and cost efficient existing methods can be applied for a more extensive coverage.

The upcoming ISO 26262 Functional Safety standard defines fault-injection testing as a relevant method to be applied for different parts of the standard. At the system level (part 4) fault-injection testing is proposed as a highly recommended method for ASIL C/D to improve test coverage of safety measures that are not invoked during normal operation. At the hardware level (part 5) fault-injection testing is also recommended for highest ASIL whenever a hardware safety mechanism is defined to analyze its response to faults. And finally, at the software level (part 6) fault-injection testing is highly recommended for ASIL C/D where arbitrary faults corrupting software or hardware components must be injected to test the safety mechanism. However the standard does not define how the existing fault-injection techniques can be applied in order to satisfy its requirements.

In this paper a simulation-based fault-injection solution based on Virtual Prototyping technology is presented. Real fault-injection scenarios are described using a Freescale dual core virtual MCU model. This work illustrates how the unmatched visibility and controllability capabilities of virtual prototypes are leveraged to build a fault-injection framework. This fault-injection framework enables the description of very complex test scenarios involving software and hardware elements. Moreover, the analysis capabilities of our solution allow tracing the software and hardware activities over time. This enables a better understanding of the correlation between fault and response and allows validating whether it is correct or not. The scripting capabilities of our framework allow fault-injection tests to be part of the overall regression testing. This removes the human interaction and reduces the effort for this type of testing dramatically.

CITATION: Reyes, V., "Virtualized Fault Injection Methods in the Context of the ISO 26262 Standard," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.* 5(1):2012, doi:10.4271/2012-01-0001.

INTRODUCTION

Automotive recalls which are due to software problems have increased exponentially over the last decade. Very recently Jaguar has announced the recall of 17.600 units only in the UK. The reasons have been hidden software glitches within the cruise control system [1]. The problem is not only the direct costs of bringing the cars to a garage and patch the software, but the indirect costs associated to brand image damage even in cases where no accidents or injuries are involved [2, 3].

In order to improve the quality of the software in a vehicle, it is not only a matter of performing more tests, but also a matter of increasing the quality of the tests. In order to increase the quality of the tests it needs to be measured whether the tests are exercising all the software functionality. That includes corner cases that are not triggered during normal operation, but which are triggered when the underlying hardware fails or the software gets corrupted. Today's safety critical systems include multiple fault tolerant mechanisms which are implemented both on hardware and software. Those mechanisms are able to detect and even

correct such faults. However, this comes with a huge impact on the overall development cost. For instance, diagnostic software takes today up to 40% of the total lines of code of an ECU and counts for up to 50% of the processor runtime [4]. Moreover, testing the diagnostic software already counts for up to 25% of the total testing costs.

Code coverage measurement and fault injection testing are two very important activities to increase the quality of the tests. However, the application of both activities during the software integration and testing phases is very limited. On the one hand, code coverage is mainly applied to on-host software unit testing only. Although useful, on-host testing at the unit test level does not exercise big parts of the embedded software running on the device. Moreover, due to the adaptations required, on-host testing is not sufficient to credibly guarantee that faulty software is not being deployed. On the other hand, conventional fault injection techniques that can be applied during software integration and test phases have multiple limitations in terms of intrusiveness, controllability and limited set of injection points that can help reaching a more extensive coverage. This will be discussed more in the next sections of the paper.

The new ISO 26262 standard [5] is imposing more severe certification requirements during the design and validation of safety critical systems. When it comes to software integration and test in particular, ISO 26262 highly recommends both fault injection testing and structural coverage metrics for the highest safety integrity levels (ASIL C/D). Due to all the extra requirements, being compliant to the new standard will obviously increase the quality of the final results, but it will also increase significantly the development and testing costs of bringing a new product to the market.

This paper discusses how to specifically apply virtual prototyping technology to enhance fault injection testing during software integration and test phases. Virtual prototype based fault injection overcomes important limitations of existing fault injection methods, such as intrusiveness, controllability and access to injection points for both permanent and transient faults.

The rest of the article is structured as follows. As part of the introduction a high level overview of the ISO 26262 standard with specific focus on its fault injection requirements is presented. After that, conventional fault injection techniques are analyzed with respect to their advantages and limitations. In the following section our proposed virtual prototype based fault injection framework is introduced in more detail. Then, we describe two fault injection scenarios and how they can be reproduced using our virtual prototype based framework. Finally, we close the paper with our conclusions.

ISO 26262 FUNCTIONAL SAFETY STANDARD

ISO 26262 is a functional safety standard that replaces the older and more generic IEC 61508 for passenger vehicles.

ISO 26262 addresses hazards caused by malfunctioning behavior of electric and electronic safety related systems. The standard provides:

- **An automotive specific safety lifecycle** - The safety lifecycle is composed of three phases: **concept** phase, **product development** phase and **after start of production** phase.

- **An automotive specific risk-based approach based on Automotive Safety Integrity Levels (or ASIL)** - Three aspects define the ranking of an item onto a specific Automotive Safety Integrity Level. **Severity**, which varies from light injuries to fatal injuries in case of failure of the item under investigation. **Exposure**, which indicates the probability that the item fails. And **controllability**, which indicates how difficult is to control the effects of the hazard by the driver. ASIL D requires the highest safety integrity level for the item and ASIL A the lowest.

- **Requirements and recommended methods for the validation of the safety levels** - The standard breaks down the requirements to verify and validate the item under-consideration on different sub-parts (system-level, hardware, software). For each of these sub-parts a set of verification methods are described on the standard, with specific weights for the different ASIL levels: no recommendation, recommended and highly recommended. For example, section 4.7 (system design) highly recommends simulation as a method to verify the system for ASIL C and D compliancy. On the other side of the V curve, section 4.8 (item integration and test) highly recommends fault injection testing for ASIL C and D compliancy.

In general ISO 26262 highly recommends simulation and prototyping methods for system, hardware and software design and verification, especially as a fault-injection technique. Fault-injection is more specifically mentioned for system, HW and SW integration and test activities. Fault-injection can be applied successfully to improve the test coverage of safety mechanisms at the system level, covering corner cases difficult to trigger during normal operation. It can also be applied whenever a hardware safety mechanism is defined to analyze its response to faults, or where arbitrary faults corrupting software or hardware components must be injected to test the safety mechanisms.

CONVENTIONAL FAULT INJECTION OVERVIEW

Fault injection helps to determine whether the response of a system matches its specification, in presence of faults. It helps understanding the effects of faults on the target system behavior, as well as assessing how efficient the existing fault tolerance mechanisms are [6].

Faults can be categorized in two big buckets: hardware faults and software faults. Hardware faults can be categorized by their duration as: **permanent faults** (triggered by component damage), **transient faults** (triggered by

environmental conditions, also known as soft-errors), and **intermittent faults** (triggered by unstable hardware). Software faults are always the consequence of an incorrect design, at the specification or at the coding time. Software faults are latent in the code and show up only during operation. Despite their permanent nature, their behaviors are transient.

Conventional fault injection methods are **hardware based fault injection**, **software based fault injection** and **simulation based fault injection** [7]. Hardware based fault injection is performed at the physical level by typically modifying the value of the pins of the Electronic Control Unit (ECU) (a.k.a. **with contact**) or by disturbing the hardware with electromagnetic interference or heavy ion radiation (a.k.a. **without contact**). Software based fault injection aims to reproduce at the software level the errors introduced by hardware faults without physically modifying the hardware. Finally, simulation based fault injection uses an implementation model at gate or RTL level to perform the experiments. Later in the paper the benefits and limitations of every method will be described in more detail.

A basic fault injection framework, as described in [8], is composed of the following parts (see Figure 1):

- Target system,
- Fault injector (that injects faults from a library),
- Workload generator (to create stimuli according to the test scenarios),
- Monitor that feed information back from the target system
- Data collector and analyzer.
- Everything orchestrated by a controller.

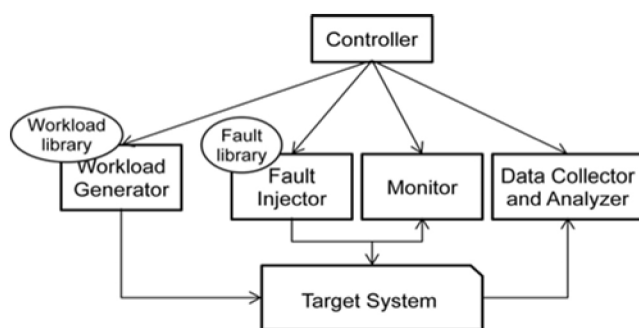


Figure 1. Conventional fault injection framework

FAULT INJECTION WITH VIRTUAL PROTOTYPES

General information about virtual prototype technology can be found in [12, 13]. In the context of this paper we will describe virtual prototypes being used as a framework to perform fault injection tests, see Figure 2. This virtual prototype based framework is composed of two main parts:

the model of the target system and the tooling around it. Obviously the model of the target system varies depending on the ECU/MCU hardware to be used, whereas the rest of the tool framework remains the same.

Virtual prototype target - A VP of the target system is a software simulation that emulates the digital hardware. In our cases it is created using the standard SystemC programming language [10] at the transactional modeling level [11]. This virtual prototype target is functionality equivalent to the actual hardware and allows running the same binary software that runs on the actual hardware. It also exposes the same external interfaces which allow connecting plant models and environment models to it. A virtual prototype provides also hooks to the tooling for inspection and modification of the internal values, something that it is not possible to do with real hardware.

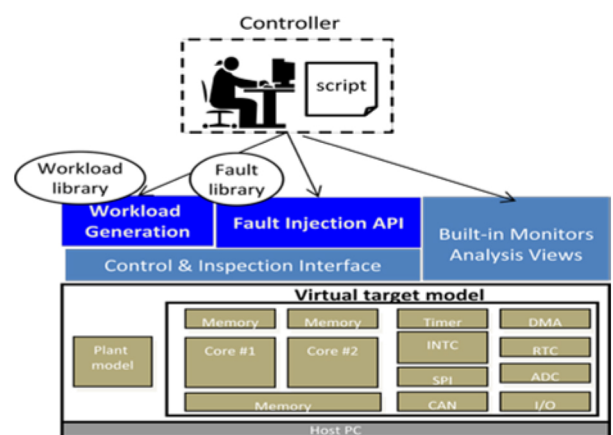


Figure 2. Virtual prototype based fault injection framework

Fault injection API - A key link of the fault injection framework to the virtual prototype target, is the generic control and inspection interface exposed by the VP. This interface allows inspecting and modifying internal and boundary values of the target system (e.g. component registers, memory content, signals, etc). This interface is also used to control the simulation execution. Based on this generic interface a higher level fault injection API is defined to inject the faults.

This API has two fundamental commands: trigger and inject. The **trigger command** invokes an injector routine when a trigger event happens. Trigger events can be software events (for example, entering a specific software routine), hardware events (for example, when an interrupt lines goes active), time events (for example, after 1 second of simulation time) or any combination of them. Triggers can be concatenated and enable other triggers dynamically based on system status. This increases the precision on when a fault can be injected. The **inject command** sets the element specified to a certain value. Supported elements are IO pins, registers, internal signals and memory locations. The value can be set just once (transient) or can be forced permanently.