The automotive industry faces a significant challenge: to release cars with zero software defects. The number of recalls and redesigns due to software problems illustrates the magnitude of the challenge—they have grown exponentially over the last decade. There are major economic implications for manufacturers since the cost of recalling a vehicle can be huge, especially when the issue affects the integrity of the brand.

Manufacturers already spend a lot of money on software testing. In fact, testing accounts for about 75% of the cost of software development. And that spend is set to grow as the number of tests that manufacturers have to run continues to increase.

But simply increasing the number of tests is not always the best way to reduce defects. Improving tests, so that they exercise corner cases that are not triggered by normal operation, improves quality.

Standards, like ISO 26262, address the planning and development of safety-critical systems and place further demands on software testing. ISO 26262 provides an automotive-specific, risk-based approach based on Automotive Safety Integrity Levels (ASIL). It specifies the requirements and recommended methods for validation of the safety levels including fault-injection testing.

Fault injection can improve test coverage of safety mechanisms (at the system level) by covering corner cases that are difficult to trigger during normal operation. It is also recommended whenever a hardware safety mechanism is defined, to analyze its response to faults, and where arbitrary faults corrupting software or hardware components must be injected to test safety mechanisms.

Fault Categories
We can categorize faults as either software or hardware faults. Within the hardware category, faults are either:
- permanent (triggered by component damage),
- transient (triggered by environmental conditions, also known as soft errors), or
- intermittent (caused by unstable hardware).

Fault-Injection Testing
Fault injection helps to determine whether the response of a system matches its specification in the presence of faults. It helps system engineers to understand the effects of faults on the target system behavior. It also helps to assess the efficiency of fault-tolerance mechanisms, and enables the design team to reduce the presence of faults during the design and implementation phases.

Virtual prototypes provide complete frameworks to create advanced fault-injection scenarios, which are non-intrusive, run faster and offer more control and visibility than traditional methods. Victor Reyes, Synopsys, explains their relevance to new safety-critical standards like ISO 26262.
Comparing Fault-Injection Techniques

Table 1 compares four traditional fault-injection techniques by considering some of the most important factors.

These include:
- the type of faults that can be triggered (fault injection points),
- the ability to model permanent faults,
- intrusiveness, or how the injection of the fault changes the original execution flow of the system,
- observability, which defines how well the set of reactions (or events) triggered by the fault can be seen and recorded,
- controllability, which defines the precision in terms of exact time and location where the fault can be injected,
- repeatability, or the ability to repeat the test in a deterministic fashion, and
- speed, which will define to some extent the complexity and duration of the test scenarios.

Typically, the test team will use ‘hardware-based with contact’ fault injection to inject errors on the IO boundary of the Electronic Control Unit, and ‘hardware-based without contact’ to trigger soft errors—for example, to simulate memory corruption due to radiation or electromagnetic interference.

The ‘with contact’ techniques can model permanent faults, and they are not intrusive (although there is a risk of damage if misused). The ‘without contact’ techniques are more focused on transient errors and are also non-intrusive. However, there is little control on when and where the fault will be injected.

Software-based fault-injection techniques can only inject errors on those locations accessible by the software, that is, memory and registers for memory-mapped peripherals, which means they are only able to model transient faults. The biggest problem with software-based fault-injection techniques is that they are intrusive: because the engineer has to modify the software binary code to inject the errors, there is a risk that the test unit may behave differently from the production software running in the field.

The techniques described above use real hardware, which limits the observability. They are controllable and repeatable, but because they use real hardware, the tests are never completely deterministic. All three techniques run fast enough (real time) to handle complex software stacks.

Simulation-based fault injection (performed at the gate or RTL level) has the advantage of having full access to all hardware elements on the system. Without being intrusive, it has full observability, controllability and determinism. However, the simulations are extremely slow, which makes them unusable on more complex fault scenarios where the design team must take the software into account.

Virtual Prototyping

A virtual prototype is a software model that emulates the hardware. Design teams can use a virtual prototype to model the digital aspects of a microcontroller unit, an electronic control unit or even a complete ECU network, and run the simulation on a desktop PC. Virtual prototypes run the same binary software as the real hardware. Because the virtual prototype is a soft model, software teams can get access to it months before the actual hardware device is available.
Virtual prototypes give development teams more than just software models for simulating the hardware; they are environments that allow them to debug and analyze hardware/software interactions. They also offer full visibility of the internal and external registers and signals, provide full control over program execution, and are completely non-intrusive.

Engineers can use virtual prototypes to freeze the full system execution at any point in time (even with multi-core hardware) and read and modify internal values. They can also use advanced analysis features to correlate software (at the application level) with hardware events, measure code coverage, apply fault injection, and use scripts to automate the simulations.

Virtual prototypes integrate seamlessly into existing software tool chains and connect to external third-party tools for hardware-in-the-loop and rest-of-bus complete simulations.

Teams can easily deploy and scale simulation models. Virtual prototypes are easy to share, archive and deploy across a worldwide organization.

**Fault Injection Using Virtual Prototyping Technology**

Virtual prototypes enable users to access all the internal hardware elements of a design—memory content, registers, signals—as well as specific, fault-tolerant mechanisms, like error correction codes (ECC) on memories, assuming, of course, that those features have been modeled. Users can create virtual prototype models without much effort by mirroring the functionality of the block at a more abstract level.

Virtual prototypes can model both transient and permanent faults. Users can inject faults through mechanisms on the simulation framework, without having to modify the embedded software models. They can also visualize and trace all hardware and software events that have been modeled on the systems. Visualization tools present both hardware and software execution and events on the same windows using the same

---

**Figure 1: Fault-injection environment**
timeline, which allows users to correlate them and see the cause-and-effect of a fault.

A basic fault-injection environment (Figure 1) includes the following elements:
- a target system—the virtual hardware model,
- a fault injector—injects faults from a library,
- the workload generator—creates stimuli according to the test scenarios
- a monitor—feeds information back from the target system and the data collector and analyzer, and
- a controller—to orchestrate everything.

The fault injector and library are based on a simple fault-injection API to model fault-injection scenarios. The API has two basic commands: trigger and inject. The trigger command invokes an injector routine when a trigger event happens. Users can concatenate triggers to enable other triggers dynamically, depending on the system status. The inject command sets the specified element 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. Users can add model-dependent commands for specific purposes (for instance, to flag an ECC error on a memory after a read/write access).

**Fault-Injection Scenario: Data Abort Due to ECC Error**

This soft-error scenario uses fault-injection to corrupt data on the SRAM memory during normal software execution. ECC functionality on the SRAM model triggers the data abort exception. The process is:
- The MCU is running an AUTOSAR software application
- Triggered by the internal interrupt line, the processor core goes into a standard, interrupt-service routine
- The first, next-access to the SRAM memory will trigger an ECC error back

Engineers can describe this scenario using the scripting facilities in the framework and the fault-injection API using just three commands and fewer than 10 lines of code.

When the core enters the software-exception routine, an error routine shuts down the OS. We can trace the fault injection and its effects using the built-in monitoring and analysis capabilities.

**Summary**

Fault injection, as recommended by the ISO 26262 standard, is the best method available to test the fault tolerance of hardware blocks and to ensure the effectiveness of diagnostic software.

Virtual prototypes provide a complete framework to create advanced fault-injection scenarios. They offer more visibility and fault-injection points than hardware-based fault injection, can model both permanent and soft errors, and, unlike software-based fault injection, virtual prototype frameworks are completely non-intrusive. Virtual prototypes run orders of magnitude faster than RTL/gate level simulators.

Fault-injection frameworks that use virtual prototypes enable users to put errors under version control and automate fault-injection regression testing every time the software changes. Virtual prototype simulations have the potential to be used as evidence for certification and compliance with standards like ISO 26262.

For more information on System-Level Solutions, visit: [http://www.synopsys.com/systems](http://www.synopsys.com/systems).

Synopsys' Virtual Prototyping solution is delivered through Virtualizer Development Kits.

For more information about the Synopsys Virtual Prototyping Solution visit us on the web at [www.synopsys.com/virtualprototypes](http://www.synopsys.com/virtualprototypes), contact your local sales representative or call 650.584.5000.