Fault-Injection testing and code coverage measurement using Virtual Prototypes on the context of the ISO 26262 standard

NMI “Automotive Electronics Systems 2013” Event

Victor Reyes
Technical Marketing
System Level Solutions
September 24th, 2013
AGENDA

Introduction
Fault-Injection with Virtual Prototypes
Code Coverage with Virtual Prototypes
Summary
The “Zero SW Defects” Challenge

• **Cost of non-quality**
  - Recalls
  - Late redesigns

• **Cost of testing**
  - Testing is already 75% of the SW development cost
  - Variants induced combinatorial complexity
    - Opel chassis: one vehicle, 72 variants, 250,000 test scenarios
    - Expensive HIL testing labs

• **Not only more testing but better testing**
  - Driven by code coverage
  - Testing when HW or SW fails
  - Diagnostic SW is already 50% of the LoC and runtime

• **Safety standards → ISO 26262**
ISO 26262 Functional Safety Standard

• **What is ISO 26262?**
  – Functional safety standard for passenger vehicles (replacing IEC 61508)
  – Addresses hazards caused by malfunctioning behavior of E/E safety related systems

• **What does it provide?**
  – Automotive safety lifecycle
  – Automotive specific risk-based approach → ASIL (Automotive Safety Integrity Levels)
  – Requirements for validation of safety levels

• **Is the standard mandatory?**
  – Not directly, but becomes “best practice” and “state-of-the-art”. This imposes legal responsibilities to manufacturers that do not apply the standard, being liable in court in case of accident.
Simulation and prototyping are highly recommended methods for System, HW and SW Design verification, specially as a fault-injection technique.

Fault-injection testing is highly recommended during Integration and Test for:
- (System) to improve test coverage of safety measures that are not invoked during normal operation
- (HW) whenever a HW safety mechanism is defined to analyze its response to faults
- (SW) where arbitrary faults corrupting software or hardware components must be injected to test the safety mechanism
The test environment for software integration testing shall correspond as closely as possible to the target environment.

- Differences can arise in the source or object code due to different bit widths of data/address words of the processors.

- Differences between the test environment and the target environment shall be analyzed in order to specify additional tests in subsequent test phases.

### Requirements for ASIL V&V (cont.)

#### 6.9 Structural coverage metrics at the SW unit level

<table>
<thead>
<tr>
<th>ASIL</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement coverage</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Branch coverage</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>MC/DC (modified condition/decision coverage)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

#### 6.10 Structural coverage metrics at the SW architecture level

<table>
<thead>
<tr>
<th>ASIL</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function coverage</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Call coverage</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>
AGENDA

Introduction

Fault-Injection with Virtual Prototypes

Code Coverage with Virtual Prototypes

Summary
Fault-Injection Overview

- **Fault-injection** help to determine whether the response of a system matches its specification, in presence of a defined set of faults. Aimed at
  - **Understanding** of the effects of the faults and thus of the related behavior of the target system
  - **Assessing** the efficacy of the fault tolerance mechanisms embedded into the target system
  - **Reducing** the presence of faults in the design/implementation of the fault tolerance systems

- **Hardware faults** (best classified by their duration)
  - **Permanent** (triggered by component damage), **Transient** (triggered by environmental conditions, a.k.a. soft-errors), **Intermittent** (unstable hardware)

- **Software faults** are always the consequence of an incorrect design, at the specification or at the coding time
  - Faults are latent in the code and show up only during operations
  - Despite their permanent nature, their behaviors are transient
## Techniques and Categorization

<table>
<thead>
<tr>
<th></th>
<th>Hardware-based Fault-Injection</th>
<th>Software-based Fault-Injection</th>
<th>Simulation-based Fault-Injection (RTL /Gate Level)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Faults injection points</strong></td>
<td>Limited set of injection points, mainly IO pin level</td>
<td>Internal (soft errors: radiation, EMI)</td>
<td>Only locations accessible by SW (memory, registers)</td>
</tr>
<tr>
<td><strong>Able to model permanent faults</strong></td>
<td>Yes</td>
<td>No</td>
<td>No (unless explicit HW support)</td>
</tr>
<tr>
<td><strong>Intrusiveness on the experiment</strong></td>
<td>None</td>
<td>None</td>
<td>High</td>
</tr>
<tr>
<td><strong>Observability</strong></td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Controllability</strong></td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td><strong>Repeatability</strong></td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Experiment speed</strong></td>
<td>Real time</td>
<td>Real time</td>
<td>Real time</td>
</tr>
</tbody>
</table>
What are Virtual Prototypes?

Fast, fully functional software model of the digital hardware executing unmodified production software and providing a higher debugging/analysis efficiency.

- Early Availability
- Easier Deployment
What are Virtual Prototypes?

Fast, fully functional software model of the digital hardware executing unmodified production software and providing a higher debugging/analysis efficiency

Visibility

Controllability

Fast

Non Intrusive

Deterministic

Scriptable

Better Developer Productivity
Where do we fit in?

Physical/mechanical models modeled in e.g. Saber, Simulink

Other ECU components (mainly mixed-signal) modeled in e.g. Saber, Simulink

Virtual Prototype (running real binary software)
# Fault-Injection using Virtual Prototypes

<table>
<thead>
<tr>
<th>Feature</th>
<th>Virtual Prototype based solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Faults injection points</strong></td>
<td>Full access to internal and external HW elements (that have been modeled), as well as SW</td>
</tr>
<tr>
<td><strong>Able to model permanent faults</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Intrusiveness on the experiment</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Observability</strong></td>
<td>High</td>
</tr>
<tr>
<td><strong>Controllability</strong></td>
<td>High</td>
</tr>
<tr>
<td><strong>Repeatability</strong></td>
<td>High</td>
</tr>
<tr>
<td><strong>Experiment speed</strong></td>
<td>1/10 real time</td>
</tr>
</tbody>
</table>

- Can modify the state of the complete system
- Faults reside in the simulation framework / do not go into release code
- All HW and SW events can be recorded and correlated
- Can be triggered by software, hardware or time events
- Completely deterministic
- Fast enough to run complete SW stacks
Virtual ECUs are Being Used by Tier1s For SW Development Compliant with ISO26262

- **Method:**
  - Injection of error into IC models by simulation backplane during program execution
- **Benefit of simulator approach:**
  - Access to failure modes, which are hard to provoke with silicon setup

Source: Advanced Fault-Injection Methods for Automotive Safety Critical System (Freescale, Continental-Teves, Synopsys)
AGENDA

Introduction

Fault-Injection with Virtual Prototypes

Code Coverage with Virtual Prototypes

Summary
Code Coverage Overview

• Code coverage help achieving **cost-effective testing**, i.e. same result with fewer tests (eliminate redundant tests)

• What to measure?
  – **Function coverage**: has each function in the SW been called?
  – **Call coverage**: has each different function call been encountered at least once?
  – **Statement coverage**: has each statement in the SW been executed?
  – **Branch coverage**: has each branch of each control structure been executed?
  – **Decision coverage**: has every decision taken all possible outcomes at least once?
  – **Condition coverage**: has each Boolean sub-expression evaluated both true and false?
  – **Modified Condition/Decision coverage**
Confidence (target) vs. Productivity (host)

• **On-target testing**
  + Minimizes the “credibility gap”, the possibility that unanticipated difference exist between execution on-host and on-target
  + Has the ability to execute all code (e.g. drivers).
  + Provides better arguments to certification authorities
    – Depends on HW resources/instrumented code

• **On-host testing**
  + Useful when target is not available or when access is limited
  + Quicker “build-run-analyze” turnarounds
    – Software requires adaptation
Coverage Flow with Virtual Prototypes

+ Good turnaround and scalability (parallel simulations)
+ Software does not require adaptations (same binary)
+ Does not consume hardware resources
+ Code does not need to be instrumented

- Few supported metrics for now (connection to external tools possible)

---

Diagram:

- Start simulation in VPA
- Enable SW Coverage
- Run test to completion
- Multiple simulations could be running in parallel
- Results file specific for a single test run
- cov-ls
- cov-report
- cov-annotate
- cov-merge
- coverage-all.xml
- Your tool here
- Overview
- HTML report
- Annotated C File

Merges individual results files into one combined results file
Example of code coverage report

- Statement and function coverage
- Analysis based on LCOV
AGENDA

Introduction
Fault-Injection with Virtual Prototypes
Code Coverage with Virtual Prototypes
Summary
Virtual Prototypes for Functional Safety

✓ Fault-injection and code coverage are two important techniques recommended by ISO 26262

✓ Virtual Prototypes provide a framework for advanced fault-injection
  o Providing more visibility and fault-injection points than HW based w/ contact FI
  o More controllable and precise to trigger soft-errors than HW based wo/ contact FI
  o Completely non-intrusive (unlike Software based FI)
  o Running orders of magnitude faster than RTL/gate level simulators

✓ Virtual Prototypes can be used to make testing more cost-effective through code-coverage measurements
  o With the turnaround and scalability of host-based methods
  o With the confidence of target-based methods

✓ „Errors“ can be put under version control and FI testing automated „and measure“ during regressions

✓ May be used as testing evidence for certification
Mazda Adopts Synopsys' Virtual Prototyping Solution for Electronic Control Unit Verification

- ECU system verification in a virtual environment.
- Shift testing on real equipment (Automobiles and HILS) to virtual environment.
- Perform dangerous live tests and difficult to reproduce conditions and scenarios
- Save development time and costs
- Enhance levels of safety, reliability and quality.

"Today, the ECU is the most important device in automobiles based on performance and cost. We need virtual prototyping not only to accelerate ECU development time while lowering cost, but also to ensure that our ECUs are safe and reliable," said Mr. Hisayoshi Naito general manager, Vehicle Development Division, Mazda.

Thank You