Formal Fault Propagation Analysis that Scales to Modern Automotive SoCs

Sergio Marchese - Jörg Grosse

www.onespin.com
Introduction to Functional Safety

The objective of functional safety:
Freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly

Functional safety risks

- Systematic Failures
  - Design errors
  - Tool errors
- Random failures
  - Hard errors
  - Soft errors

Risk drivers

- Continuous increase in flow and tool complexity
- Continuous increase in functionality
- Increasing density of the design process node
- Decreasing energy levels

Risk management through functional safety standards

- Minimize systematic errors
- Safeguard against random errors
ISO 26262 covers E/E systems for road vehicles
## Systematic and Random HW Faults

<table>
<thead>
<tr>
<th>Systematic SW</th>
<th>Random HW</th>
<th>Systematic HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mistake in loop termination condition</td>
<td>Radiation causes bit-flip in memory</td>
<td>Incorrect logic due to mistake in RTL model</td>
</tr>
<tr>
<td>Wrong results in processing of sensor data</td>
<td>Untimely activation of ignition coil</td>
<td>Engine control unit interrupts operation</td>
</tr>
</tbody>
</table>

**Focus:** Safety Mechanisms and Diagnostic Coverage

**Minimize Systematic Faults**
- Rigorous Verification
- Quantification of Verification

**Safeguard Against Random Faults**
- Verification of Safety Mechanisms
- Diagnostic Coverage

**Required by Safety Standards ISO 26262**
Single Point Fault Metric

Safe faults
- Not in safety relevant parts of the logic
- In safety relevant logic but unable to impact the design function (cannot violate a safety goal)

Single point faults
- Dangerous, can violate the safety goal and no safety mechanism

Residual faults
- Dangerous, can violate the safety goal and escape the safety mechanism

Multipoint faults
- Can violate the safety goal but are observed by a safety mechanism
- Sub-classified as “detected”, “perceived” or “latent”

- Unidentified safe faults must be considered residual
- Lower diagnostic coverage

Diagram: Courtesy International Standards Organization (ISO)
Diagnostic Coverage of Safety Mechanisms

**Challenges**

- Diverse/complex safety mechanisms
- High number of faults
- Final metrics computed on gate-level netlist
- Fault simulation scales but fault coverage often insufficient
- Expert judgment of coverage holes time consuming and error prone
- Formal could help but suffer from complexity issues
Safe Stuck-at Faults

Safe faults due to static IC operation modes
- Debug mode disabled
- Test logic

Explicit redundancy in hardware masks the effect of fault
- Performance impact only
- States never used in safe operation mode

Truly redundant logic such as synthesis deficiencies

Safety unrelated logic
- Design parts which do not impact the safety goal

Safe faults cannot propagate to observation points
- Mission outputs
- Internal registers

Inputs to Formal FPA
- Design (RTL/Gates)
- Constraints
- Fault population (optional)
- Observation points
Split Formal Analysis into Fast and Deep

Fast Formal Fault Propagation Analysis

- Select appropriate proof strategies targeting
  - Safe faults only (hold results rather than fail)
  - Large fault populations
- Drop debug information to gain capacity/speed
- Drop hard faults ASAP to avoid wasted effort
- Smart clustering of faults
- Automatically orchestrate mix of methods
  - Cone of influence analysis
  - Combinatorial/Sequential Equivalence Checking
  - Auto-generated assertions

Goal
- Find the majority of safe faults as fast as possible
Split Formal Analysis into Fast and Deep

Deep Formal Fault Propagation Analysis

- Select appropriate proof strategies targeting
  - Hard safe faults
  - Propagatable faults
  - Small fault populations
- Debug information is crucial
- Additional user input (e.g. SVA constraints)

Goals

- Find additional safe faults
- Debug propagatable faults
Integration of Formal FPA with Simulation

• Two-mode approach fits well with simulation flow
Case Studies Results

Basic constraints
- Debug off
- Test off

<table>
<thead>
<tr>
<th>FAST FPA</th>
<th>Identify Safe Stuck 0/1 Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTL</td>
<td>Gate-level</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Open Core Processor – 10K LoC – 794 FFs</td>
</tr>
<tr>
<td><strong>Fault Population</strong></td>
<td>12,260 faults</td>
</tr>
<tr>
<td><strong>CPU Time</strong></td>
<td>2 minutes</td>
</tr>
<tr>
<td><strong>Safe Faults</strong></td>
<td>3257 (26%)</td>
</tr>
</tbody>
</table>

- Experience demonstrates that designs have a significant number of safe faults …
- … that can be unveiled with fast targeted formal analysis
Case Studies Results

Basic constraints
• Debug off
• Test off

- Deep analysis does find additional safe faults
- For certain designs engineers want to examine every fault
- Debug and integration with SVA crucial to examine unexpected propagatable faults
Analysis of Propagatable Faults

1. Deep FPA
2. Fault Propagates? (no)
3. Add Constraint
4. Debug Trace
5. Constraint Missing? (no)
6. Done
7. Constraint Missing? (yes)
Propagatable Fault Debug Trace
Questions?