How to Stay Out of the News with ISO26262-Compliant Verification

Charles Battikha (chuck_battikha@mentor.com)
Doug Smith (doug_smith@mentor.com)
Agenda

• Taking New Products into the Automotive Market…
  Welcome to Functional Safety

• From Analysis to Fault Campaigns

• Break

• How Formal Reduces Fault Analysis for ISO 26262

• Requirement Tracing in the ISO26262 World
Entering the Automotive Market…

- Reviewing the challenges and requirements in the Automotive Market & ISO26262 Standard
Automotive Market Drivers

- Electrification
- Smart Sensors
- Sensor Fusion
- Vehicle Networking
- V2X Connectivity
- ADAS

**The Autonomous Car**

High-end cars will contain more than $6,000 worth of electronics in five years, driving a $160 billion automotive electronics market in 2022”

Luca De Ambroggi
Principal analyst Automotive electronics
IHS Markit
Complexity of Automotive Systems

**LINES OF CODE**
- Hubble Space Telescope: 5M
- Mars Curiosity Rover: 12M
- Smartphone OS: 25M
- F=35 Fighter Jet: 100M

**Source:** Roland Berger, CarsGuide

- 6 radar beams
- 8 cameras
- 12 parking sensors
- 144 electronic control units
- 500 LEDs
- 734 wire harnesses
- 2,400 wires
- 5,000 meters of cables
**Functional Safety = Table Stakes**

The goal is to create a highly assured design by removing unreasonable risk. ASIL D is set by the OEM.

<table>
<thead>
<tr>
<th>Severity of Injuries</th>
<th>Probability Exposure</th>
<th>Driver Control</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>E1</td>
<td>QM</td>
<td>QM</td>
</tr>
<tr>
<td>E2</td>
<td>QM</td>
<td>QM</td>
<td>QM</td>
</tr>
<tr>
<td>E3</td>
<td>QM</td>
<td>QM</td>
<td>A</td>
</tr>
<tr>
<td>E4</td>
<td>QM</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>S2</td>
<td>E1</td>
<td>QM</td>
<td>QM</td>
</tr>
<tr>
<td>E2</td>
<td>QM</td>
<td>QM</td>
<td>A</td>
</tr>
<tr>
<td>E3</td>
<td>QM</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>E4</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>S3</td>
<td>E1</td>
<td>QM</td>
<td>QM</td>
</tr>
<tr>
<td>E2</td>
<td>QM</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>E3</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>E4</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

**Severity of Injuries**
- S1: Life-threatening, fatal injuries
- S2: High probability of exposure
- S3: Difficult for driver to control

**E1-E4**
- E1: QM (Qualified Management)
- E2: QM (Qualified Management)
- E3: A (Approved)
- E4: B (Better)

Goal is the create a **highly assured** design by removing **unreasonable risk**.
Functional Safety Tradeoffs

IC/IP with ISO Compliance and higher ASIL ratings reduces effort, cost, and complexity for integrators.

Integrator Cost (Components)
Integrator Development Effort

IP / IC Cost (Area)
IP / IC Power (Area)
IP / DC Development Effort
Increased ASIL

Increased Cost & Development Examples:
- Reviews, Audits, Assessments
- Additional Logic in Safety Mechanisms
- Additional Rigor & Deliverables

ASIL Decomposition

Redundancy creates:
- Complexity
- Verification effort
- Area & cost penalty
ISO 26262 allows **tailoring** to match project specifics to the standard.
ISO26262 – The known

Established IC/IP Developers likely have strong development & verification processes:

- Development process well documented
- … & shown to be followed
- Create and maintain artifacts
- Requirements tracing
- Source Control
- Change Management
- Documentation Control
ISO26262 – The unknown

IC/IP developers new to the automotive market will find:

- Safety Culture
- Safety Requirements
- Safety Mechanisms
- Safety Analysis
- FMEA / FMEDA / DFA / FTA
- Fault Metrics
- Fault Insertion Campaign
- Safety Manual
- Tool Qualification
Requires Two Testing Approaches

Systematic Failures
Introduced in product development

- Incorrect Requirements
- Inaccurate/incomplete specs
- RTL Errors
- Timing Errors

Random Failures
Introduced by the environment

- Vibration
- Moisture/Dirt
- Noise
- EMI
- Electro-migration
## Fault Types

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th><strong>Safe Faults</strong>; Does not effect the Safety Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_{SPF}$</td>
<td><strong>Single Point Fault</strong>; Fault violating a Safety Requirements. Not covered by a Safety Mechanism. Should be addressed.</td>
</tr>
<tr>
<td>$\lambda_{RF}$</td>
<td><strong>Residual Faults</strong>; Faults not detected by an intended Safety Mechanism and lead to a violation of Safety Requirements. Single Point Faults and Residual Fault are not differentiated from a fault analysis perspective. <strong>Diagnostic Coverage</strong> measures effectiveness of safety mechanism in detecting Residual Faults – permanent and transient.</td>
</tr>
</tbody>
</table>

$\lambda_{SPF} + \lambda_{RF} \Rightarrow \text{SPFM (ASIL Goal)}$
## Fault Types

<table>
<thead>
<tr>
<th>( \lambda_{\text{DPF,DP}} )</th>
<th><strong>Dual-Point Faults – Detected/Perceived</strong>; Combination of independent faults that may lead to a violation of Safety requirements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_{\text{DPF,L}} )</td>
<td><strong>Dual-Point Faults – Latent</strong>; Faults not detected by safety mechanisms that would lead to a dual-point failure. Considered to be a fault in primary safety mechanism that is undetectable.</td>
</tr>
</tbody>
</table>

![Fault Diagram](image)

- **Safety Mechanism**
  - Fault in Primary Safety Mechanism not found by HW/SW BIST
- **Safety Related Function**
  - Detection of Fault 2 masked by Fault 1
- **Detection Time Interval**
  - Time

\( \lambda_{\text{DPF,L}} \rightarrow \text{LFM (ASIL Goal)} \)
Diagnostic Coverage

DC: Proportion of the failure rate that is detected or controlled by implemented safety mechanisms.

\[ \text{DC}_{RF} = \left(1 - \frac{F_0 + F_1 + F_2 + F_3}{F_0}\right) \times 100 \]

\[ \text{DC}_{DPF, L} = \left(1 - \frac{F_0 + F_1 + F_2 + F_3}{F_0}\right) \times 100 \]

Safety Mechanism

<table>
<thead>
<tr>
<th>Safety Output</th>
<th>Does Not Detect Fault</th>
<th>Detects Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Affected</td>
<td>( F_1 ) ( F_{SAFE} )</td>
<td>( F_2 ) ( F_{SAFE} )</td>
</tr>
<tr>
<td>Affected</td>
<td>( F_0 ) ( F_{RF} )</td>
<td>( F_3 ) ( F_{SAFE,DET} )</td>
</tr>
</tbody>
</table>
Verification of Random Hardware Faults

Model Random Faults

Verify Safety Mechanism

Measure Effects of Random Faults

Design Hardening

Report Metrics

- SPFM, LMF, PMHF

Meet ASIL Requirements? Meet minimum DC?

Safe?

Diagnostic Coverage

Compare against FMEDA

- Power On Self Test
- Interrupt-driven BIST
- ECC
- CRC / Checksums
- Watchdog Timers
- Voting
- Main versus Redundant Compare
- Software/Firmware Diagnostics
- Etc.
POST & Interrupt Driven BIST as SM

Normal State Flow

Run BIST

Stop Operations
Save State
Logic BIST Execution
Reporting
Restore State
Resume Operations

FTTI determines:
1. If POST only sufficient
2. Frequency of BIST

Power Off
“Key On”
Run POST
POST Pass
POST Fail
Run BIST
Safe State
Interrupt or at Prescribed Time
BIST Fail
BIST Pass
Normal Operations
Resume Operations
Mentor Functional Safety Process

Analysis
- Fault Planning
- Pruning

Design Improvement
- Structural Analysis
- Fault Analysis

UCDB
- Create Metrics
- Assumptions / Requirements Rationales
- Safety Manual

Fault Analysis
- Failure Analysis (P)
- Failure Analysis (T)

Simulation
- Simulation Fault Campaign
- Simulation Campaign

Test Profiling
- Test Profiling

Emulation
- Emulation Fault Campaign
- Emulation Campaign

SPFM, LFM, PMHF
DC per SM
A1 = combination($F_{RF} F_{SAFE}$)

A2 = combination($F_{DP,DET} F_{RF} F_{SAFE}$)

Likely residual fault distribution:

$F_{RF,A1} >> F_{RF,A2}$

Creates a max ceiling for DC.

Goal: Reduce area of A1 before starting fault campaign.
In an IC, common DFI:
- Clock Distribution
- Reset Distribution
- Power Distribution
- Main Data Busses

But easy in an IC to create DFI...
DFA & COI

• Use COI to find unintended overlap which implies shared resources
• Cutpoints & black-boxes stop COI tracing when function is protected
Summary

• ISO26262 *is* Functional Safety

• Requires many companies to create a Safety Culture

• Requires strong development and verification processes

• Requires analysis to address random hardware faults

• Reaching higher ASIL ratings will increase effort and costs
From Analysis to Fault Campaigns

Charles Battikha (chuck_battikha@mentor.com)
Topics

• Recap of Safety Analysis

• Usage of Metrics

• Analysis

• Fault Injection Campaign

• Summary
Safety Analysis

• Qualitative Analysis
  – Effects & Causes → FMEA, FTA
  – Dependent Failure Analysis
• Quantitative Analysis
  – Metrics → FMEDA, FTA
  – Analysis of Random Faults
• Fault Injection Testing
  – Verification of Safety Mechanisms
  – Metrics
Usage of Metrics

- PMHF – Targets distributed top-down
- SPFM/LFM – Bottom-up, abstracts details of diagnostic coverage within the IC/IP
FMEA & FTA

• Analysis Process to Identify
  – Failure Modes in a function
  – Effects of the failure
  – Potential Causes of the failure

• Information allows definition of
  – Safety Mechanisms
  – Reaction to failure / Safe States
  – Safety Requirements

• FMEA versus FTA versus FMEDA
### FMEDA – General Structure

<table>
<thead>
<tr>
<th>#</th>
<th>Block</th>
<th>Safety Related Element (Y/N)?</th>
<th>Failure Mode</th>
<th>Effect of Failure Mode</th>
<th>λ (FIT)</th>
<th>Potential to violate a Safety Goal in absence of safety mechanism (Y/N)?</th>
<th>Is there a safety mechanism in place to control failure mode (Y/N)?</th>
<th>Safety Mechanism(s) allowing the system to prevent the failure mode from violating the safety goals (e.g. SM1, SM2)</th>
<th>Failure mode (diagnostic) coverage (%)</th>
<th>λ&lt;sub&gt;SPF&lt;/sub&gt;</th>
<th>λ&lt;sub&gt;RF&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary Bridge</td>
<td>Y</td>
<td>Incorrect data written into transmit or configuration register(s)</td>
<td>15%</td>
<td>Incorrect or no SPI Transmission</td>
<td>3.50</td>
<td>Y</td>
<td>Y</td>
<td>SM5</td>
<td>80%</td>
<td>0.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>O</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
<th>T</th>
<th>U</th>
<th>Potential to violate a SG, in combination w/ one other independent failure (Y/N)?</th>
<th>Is there a safety mechanism in place to control latent faults (Y/N)?</th>
<th>Safety mechanism(s) allowing to prevent the failure mode from being latent ?</th>
<th>Failure mode (diagnostic) coverage w latent failures</th>
<th>λ&lt;sub&gt;SAFE&lt;/sub&gt;</th>
<th>λ&lt;sub&gt;MP,F&lt;/sub&gt;</th>
<th>λ&lt;sub&gt;MPF,DP&lt;/sub&gt;</th>
<th>Justification / Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>SM3</td>
<td>70%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>SM3</td>
<td>70%</td>
<td>0.000</td>
<td>0.126</td>
<td>0.294</td>
<td></td>
</tr>
</tbody>
</table>
**FMEDA**

Only Safety Related Elements

Distribution of Failure Modes = Engineering Judgment

\( \lambda \) (lambda) = Determined by the IC technology, distributed by area/transistor count to each block

<table>
<thead>
<tr>
<th>#</th>
<th>Block</th>
<th>Safety Related Element (Y/N)?</th>
<th>Failure Mode</th>
<th>Failure Mode Ratio (%)</th>
<th>Effect of Failure Mode</th>
<th>Potential to violate a Safety Goal in absence of safety mechanism (Y/N)?</th>
<th>Is there a safety mechanism in place to control failure mode (Y/N)?</th>
<th>Safety Mechanism(s) allowing the system to prevent the failure mode from violating the safety goals (e.g., SM1, SM2)</th>
<th>Failure mode (diagnostic) coverage (%)</th>
<th>( \lambda_{SPF} )</th>
<th>( \lambda_{RF} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary Bridge</td>
<td>Y</td>
<td>Incorrect data written into transmit or configuration register(s)</td>
<td>-15%</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>SM5</td>
<td>80%</td>
<td>0.000</td>
</tr>
</tbody>
</table>
**FMEDA - SPF**

<table>
<thead>
<tr>
<th>#</th>
<th>Block</th>
<th>Safety Related Element (Y/N)?</th>
<th>Failure Mode</th>
<th>Failure Mode Ratio (%)</th>
<th>Effect of Failure Mode</th>
<th>Potential to violate a Safety Goal in absence of safety mechanism (Y/N)?</th>
<th>Is there a safety mechanism in place to control failure mode (Y/N)?</th>
<th>Safety Mechanism(s) allowing the system to prevent the failure mode from violating the safety goals (e.g. SM1, SM2)</th>
<th>Failure mode (diagnostic) coverage (%)</th>
<th>$\lambda_{SPF}$</th>
<th>$\lambda_{RF}$</th>
<th>$\lambda_{MPF}$</th>
<th>$\lambda_{SAFE}$</th>
<th>$\lambda_{MP,L}$</th>
<th>$\lambda_{MP,F,DP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary Bridge</td>
<td>Y</td>
<td>Incorrect data written into transmit or configuration register(s)</td>
<td>15%</td>
<td>Y</td>
<td>Incorrect or no SPI Transmission</td>
<td>Y</td>
<td>SM5</td>
<td>80%</td>
<td>0.000</td>
<td>0.105</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Equations derived from ISO 26262:2018 Part 5 Annex B**

**Implication: SPF > RF > MPF**

Single Point Failure Rate = Failure Mode % * Block’s Lambda
**FMEDA - RF**

Residual Failure Rate

\[ \text{Residual Failure Rate} = \text{Failure Mode \%} \times \text{Block's Lambda} \times (1 - \text{Safety Mechanism Diagnostic Coverage}) \]

<table>
<thead>
<tr>
<th>#</th>
<th>Block</th>
<th>Safety Related Element (Y/N)?</th>
<th>Failure Mode</th>
<th>Failure Mode Ratio (%)</th>
<th>Effect of Failure Mode</th>
<th>( \lambda ) (FIT)</th>
<th>Potential to violate a Safety Goal in absence of safety mechanism (Y/N)?</th>
<th>Is there a safety mechanism in place to control failure mode (Y/N)?</th>
<th>Safety Mechanism(s) allowing the system to prevent the failure mode from violating the safety goals (e.g., SM1, SM2)</th>
<th>Failure mode (diagnostic) coverage (%)</th>
<th>( \lambda_{SPF} )</th>
<th>( \lambda_{RF} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary Bridge</td>
<td>Y</td>
<td>Incorrect data written into transmit or configuration register(s)</td>
<td>25%</td>
<td>Incorrect or no SPI Transmission</td>
<td>3.50</td>
<td>Y</td>
<td>Y</td>
<td>SM5</td>
<td>80%</td>
<td>0.000</td>
<td>0.105</td>
</tr>
</tbody>
</table>

**Potential to violate a SG, in combination w/ one other independent failure (Y/N)?**

**Is there a safety mechanism in place to control latent faults (Y/N)?**

**Safety mechanism(s) allowing to prevent the failure mode from being latent?**

**Failure mode (diagnostic) coverage w latent failures**

<table>
<thead>
<tr>
<th>Potential to violate a SG, in combination w/ one other independent failure (Y/N)?</th>
<th>Is there a safety mechanism in place to control latent faults (Y/N)?</th>
<th>Safety mechanism(s) allowing to prevent the failure mode from being latent?</th>
<th>Failure mode (diagnostic) coverage w latent failures</th>
<th>( \lambda_{SAFE} )</th>
<th>( \lambda_{MP,L} )</th>
<th>( \lambda_{MPF,DP} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>SM3</td>
<td>70%</td>
<td>0.000</td>
<td>0.126</td>
</tr>
</tbody>
</table>
### FMEDA - Safe

Safe Failure Rate
= Case 1: Failure Mode % * Block’s Lambda
= Case 2: Failure Mode % * Block’s Lambda
* Safety Mechanism Diagnostic Coverage

<table>
<thead>
<tr>
<th>#</th>
<th>Block</th>
<th>Safety Related Element (Y/N)?</th>
<th>Failure Mode</th>
<th>Failure Mode Ratio (%)</th>
<th>Effect of Failure Mode</th>
<th>(\lambda_{(FIT)})</th>
<th>Potential to violate a Safety Goal in absence of safety mechanism (Y/N)?</th>
<th>Is there a safety mechanism in place to control failure mode (Y/N)?</th>
<th>Safety Mechanism(s) allowing the system to prevent the failure mode from violating the safety goals (e.g., SM1, SM2)</th>
<th>Failure mode (diagnostic) coverage (%)</th>
<th>(\lambda_{SPF})</th>
<th>(\lambda_{RF})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary Bridge</td>
<td>Y</td>
<td>Incorrect data written into transmit or configuration register(s)</td>
<td>15%</td>
<td>Incorrect or no SPI Transmission</td>
<td>3.50</td>
<td>Y</td>
<td>Y</td>
<td>SM5</td>
<td>80%</td>
<td>0.000</td>
<td>0.105</td>
</tr>
</tbody>
</table>
### FMEDA - Safe

The Safe Failure Rate calculation is given by:

\[ \text{Safe Failure Rate} = \begin{cases} 
\text{Case 1: Failure Mode} \% \times \text{Block's Lambda} \\
\text{Case 2: Failure Mode} \% \times \text{Block's Lambda} \\
\times \text{Safety Mechanism Diagnostic Coverage}
\end{cases} \]

<table>
<thead>
<tr>
<th>#</th>
<th>Block</th>
<th>Safety Related Element (Y/N)?</th>
<th>Failure Mode</th>
<th>Failure Mode Ratio (%)</th>
<th>Effect of Failure Mode</th>
<th>( \lambda ) (#FIT)</th>
<th>Potential to violate a Safety Goal in absence of safety mechanism (Y/N)?</th>
<th>Is there a safety mechanism in place to control failure mode (Y/N)?</th>
<th>Safety Mechanism(s) allowing the system to prevent the failure mode from violating the safety goals (e.g. SM1, SM2)</th>
<th>Failure mode (diagnostic) coverage (%)</th>
<th>( \lambda_{SPF} )</th>
<th>( \lambda_{RF} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary Bridge</td>
<td>Y</td>
<td>Incorrect data written into transmit or configuration register(s)</td>
<td>15%</td>
<td>Incorrect or no SPI Transmission</td>
<td>3.50</td>
<td>Y</td>
<td>Y</td>
<td>SM3</td>
<td>80%</td>
<td>0.000</td>
<td>0.105</td>
</tr>
</tbody>
</table>

**Case 2 – Detected with a no multi-point failure potential**

Potential to violate a SG, in combination with one other independent failure (Y/N)?

Is there a safety mechanism in place to control latent faults (Y/N)?

Safety mechanism(s) allowing to prevent the failure mode from being latent?

Failure mode (diagnostic) coverage w latent failures

\( \lambda_{SAFE} \)

\( \lambda_{MP,L} \)

\( \lambda_{MP,F,DP} \)

Justification / Rationale

\[ \lambda_{SAFE} = 0.000 \]

\[ \lambda_{MP,L} = 0.126 \]

\[ \lambda_{MP,F,DP} = 0.294 \]
**FMEDA – MPF**

MPF, Detected Failure Rate = Failure Mode % * Block’s Lambda * Diagnostic Coverage of both Safety Mechanisms.

<table>
<thead>
<tr>
<th>#</th>
<th>Block</th>
<th>Failure Mode</th>
<th>Failure Mode Ratio (%)</th>
<th>Effect of Failure Mode</th>
<th>λ (FIT)</th>
<th>Potential to violate a Safety Goal in absence of safety mechanism (Y/N)?</th>
<th>Is there a safety mechanism in place to control failure mode (Y/N)?</th>
<th>Safety Mechanism(s) allowing the system to prevent the failure mode from violating the safety goals (e.g., SM1, SM2)</th>
<th>Failure mode (diagnostic) coverage (%)</th>
<th>λ_{SPF}</th>
<th>λ_{RF}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary Bridge</td>
<td>Incorrect data written into transmit or configuration register(s)</td>
<td>15%</td>
<td>Incorrect or no SPI Transmission</td>
<td>3.50</td>
<td>Y</td>
<td>Y</td>
<td>SM5</td>
<td>80%</td>
<td>0.000</td>
<td>0.105</td>
</tr>
</tbody>
</table>

Potential to violate a SG, in combination w/ one other independent failure (Y/N)?
Is there a safety mechanism in place to control latent faults (Y/N)?
Safety mechanism(s) allowing to prevent the failure mode from being latent?
Failure mode (diagnostic) coverage w/ latent failures

<table>
<thead>
<tr>
<th>λ_{SAFE}</th>
<th>λ_{MP,L}</th>
<th>λ_{MPF,DP}</th>
<th>Justification / Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.126</td>
<td>0.294</td>
<td></td>
</tr>
</tbody>
</table>
Where does Diagnostic Coverage come from?

Answer: ISO 26262 Part 5 Annex D & Part 11 OR Fault Campaign OR Expert Judgement

Safety Mechanisms that are standard / well understood can rely solely on the standard / documented sources. Position tends to vary with customers.
Layered Fault Campaign

Fault Pruning

RF
MPF,L

Safe
No

Fault propagates to function?

RF
MPF,L

No

Fault Analysis

Failure
Analysis

Structural Analysis

Cones of Influence (COI)

Formal Analyze
Undetected Failures

Simulation

Emulation

Safe
SPF
RF
DPF,DP
DPF,L

RF
DPF,L

Calculate MSD

FDI≤

MPF,L

DPF,DP propagates to SM?
Increase Confidence

Failure Modes

- Stuck-at 0
- Stuck-at 1
- Transients

Diagnostic Coverage

Worst Case
Most Pessimistic

Structural Analysis

Categorize based on cones of influence

Safe
SPF
RF
DPF,DP
DPF,L

Stuck-at 0
Stuck-at 1
(No transients)

Calculate Minimum Sequential Distance

MSD > FDI
RF
DPF,L

Fault Analysis

DPF,DP propagates to SM?

No
RF
DPF,L

Yes
DFP,L

Failure Analysis

Undetectable DPF,DP Failures?

No
RF
DPF,L

Yes
DFP,L

Fault propagates to function?

No
Safe

Yes
Undetectable Fault Failures?

Yes
RF

Worst Case
Most Pessimistic

Better
Less Pessimistic

Highest Confidence
Least Pessimistic
Design Hardening

- Beyond providing/validation of Metrics, Fault Campaigns provide
  - Verification of safety mechanisms
  - Insight into improving coverage

- Need insight into where faults fall
Aggregating / Mapping Coverage

FMEDA generates SPFM, LFM, & PMHF (est.)

<table>
<thead>
<tr>
<th>Design Block</th>
<th>DC of SM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1</td>
<td>X%</td>
</tr>
<tr>
<td>Block 2</td>
<td>Y%</td>
</tr>
<tr>
<td>Block 3</td>
<td>Z%</td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
</tr>
</tbody>
</table>

Fault Campaign ran on Blocks A, B, D – individually or grouped – based on Function / SM

Campaign collects across each block, the sum of:

- $F_{SPF}$
- $F_{RF}$
- $F_{SAFE}$
- $F_{DP,DET}$
- $F_{DP,LAT}$

Aggregates Faults to create Diagnostic Coverage for Block 1:

- $DC_{RF,Block1}$
- $DC_{LF,Block1}$

Aggregated Diagnostic Coverage

Compare

Block 1 (Safety Related, ASIL)

Sub-block A
- Function
- PSM
- SSM

Sub-block B
- Function
- PSM
- SSM

Sub-block C
- Non-Safety Function
- PSM
- SSM

Sub-block D
- Function
- PSM
- SSM
Management & Tracing

• Challenges:
  – Managing the Size/Complexity of FMEDA Spreadsheet
  – Supporting internal reviews and audits
  – Supporting external assessments

Fault Campaign  How to Connect & Manage?  FMEDA
Fault campaign process (1)

FMEDA

Tracking

Back-annotate

Safety Info

Fault List

Testbenches

UCDB

Diagnostic Coverage

UCDB

Simulation/Emulation Fault Campaign

Fault Campaign

FDI
Fault campaign process (2)

Safety Definitions

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Description</th>
<th>Safety Path Expression</th>
<th>Primary Safety Mechanism Expression</th>
<th>Secondary Safety Mechanism Expression</th>
<th>Fault Detection Time</th>
<th>Multi-Point Fault Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>SR-1</td>
<td>Top Module</td>
<td></td>
<td>ip_error</td>
<td>ip_error</td>
<td>5</td>
<td>5 TEST_ATTRIB</td>
</tr>
<tr>
<td>1.1.1</td>
<td>SR-1.1</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>dist_a &amp; (stb &amp; cyc)</td>
<td>ip_error</td>
<td>ip_error</td>
<td>5</td>
<td>5 TEST_ATTRIB</td>
</tr>
<tr>
<td>1.1.2</td>
<td>SR-1.2</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>act &amp; (stb &amp; cyc)</td>
<td>ip_error</td>
<td>ip_error</td>
<td>5</td>
<td>5 TEST_ATTRIB</td>
</tr>
<tr>
<td>1.1.3</td>
<td>SR-1.3</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>interrupt</td>
<td>ip_error</td>
<td>ip_error</td>
<td>5</td>
<td>5 TEST_ATTRIB</td>
</tr>
<tr>
<td>1.4.1</td>
<td>SR-1.4</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>scik</td>
<td>ip_error</td>
<td>ip_error</td>
<td>5</td>
<td>5 TEST_ATTRIB</td>
</tr>
<tr>
<td>1.5</td>
<td>SR-1.5</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>mosi</td>
<td>ip_error</td>
<td>ip_error</td>
<td>5</td>
<td>5 TEST_ATTRIB</td>
</tr>
<tr>
<td>1.6</td>
<td>SR-1.6</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td></td>
<td>ip_error</td>
<td>ip_error</td>
<td>5</td>
<td>5 TEST_ATTRIB</td>
</tr>
</tbody>
</table>

Fault Campaigns

Safety Info

Diagnostic Coverage

UCDB

Template + Annotate

Tracking

Verification Management Tracker

FMEDA
Summary

• Qualitative Analysis with a FMEA/FTA & Quantitative Analysis with a FMEDA are standard practices

• FMEDA is a key document to allows integrators of IC/IP to understand functional safety metrics
  – Especially important when considering configuration / feature options

• Connecting information from Fault Injection Campaigns to the FMEDA
  – Validates early predictions of Diagnostic Coverage and Hardware Architectural Metrics
  – With challenging architectures, the only means to determine coverage

• Fault Injection Campaigns serve as verification of safety mechanisms
Break
How Formal Reduces Fault Analysis for ISO 26262

Doug Smith
Doug_Smith@mentor.com
Verification Consultant
Mentor Consulting
Safety on semiconductors

Focus on Safety Element

Traditional

Safety Element

Safety Mechanism

Design

Safety Sub-element

Safety Mechanism

Safety Sub-element

Safety Mechanism

Safety Sub-element

Safety Mechanism

The design is the safety element!
ICs are harder

- Potentially lots of
  - Safety critical functions
  - Safety mechanisms
  - Secondary safety mechanisms
- Large designs → thousands of random faults to inject!
- How to categorize faults shared between shared logic?
- Need tests that allow faults to propagate and be detected
- Large simulation time to test software safety mechanisms
- May have large fault detection time intervals
Try breaking up the problem!

- Not allowed 🙁
- Must show independence with Dependent Fault Analysis (DFA)

Solution – Formal cones of influence
Formal cone of influence (COI)

- Cone is equivalent to DFA
- Cones can overlap
- Cones enable quick fault categorizing
COI fault analysis

- Safe fault
- Single-point fault
- Residual fault
- Dual-point fault in safety function
- Dual-point fault in safety mechanism
- Latent fault
Minimum sequential distance (MSD)

- Fault Detection Interval (FDI)
- Violation if \( MSD_{SM} - MSD_{SF} > FDI \)

E.g.,

\[ \text{FDI} = 2 \]
\[ 6 - 3 > 2 \]

\[ \therefore \text{Too long to propagate} \ldots \text{safety goal violation!} \]
Safety function fault propagation

No propagation -> Safe fault!
Safety mechanism fault propagation

No propagation -> undetectable fault!

Where does the stimulus come from?
Traditional formal

• Input constraints and assumptions

```
asm_drive_data    : assume property ( pkt_val |-> pkt_data == data );
asm_pkt_stable    : assume property ( pkt_val |-> $stable(packet) );
asm_payload_stable: assume property ( pkt_val |-> $stable(payload) );
asm_pkt_kind_stable: assume property ( pkt_val |-> $stable(pkt_type) );
```

• Issues
  – Need input requirements
  – Labor intensive
  – Not automated
  – Typically incomplete – formal tries everything!

Solution - SLEC
Sequential Equivalency Checking

Original design

Fault-injected design

Tie inputs together
Injecting faults

Fault condition

Driver value

netlist cutpoint dut.reg_o -cond { fault[100] } -driver 1'b0
Formal fault injector

module fault_injector(...);
    default clocking cb @(glob=2$al\_clock); endclocking
    bit [N:0] fault = '0;

    asm_single_point_fault: assume property ( $onehot0 (fault) );
    asm_fault_stuckat: assume property ( fault |=> $stable(fault) );

• Conditional cutpoint

netlist cutpoint {impl.dut.tx_data_fifo0.fifo0.genblk2.mem0.rdata[2]} \ 
    -cond {impl.fi.fault[1]} -driver 1'b0

• SLEC target

slec map spec.safecheck.safety0 \ 
    impl.safecheck.safety0 -cond { impl.safecheck.fi.fault[1] }
Parallel fault analysis

- One compile
- Thousands of parallel fault targets analyzed by formal
Proving a fault propagates

SLEC

Mismatch proves fault propagates

Fault injected
First steps …

- Quick and easy
- But do all faults really propagate?

Structural analysis

Fault analysis
Need failure analysis …

• Simultaneous propagation to output and safety mechanism?
  – Within time window?

Failure analysis

```c
// Find residual faults
cover property ( fault && seq_fault_propagates |-> seq_fault_not_detected[*FDI] );

// Find latent faults
cover property ( fault && seq_psm_fault_propagates |->
  seq_ssm_fault_not_detected[*MPFDI] );
```
Example undetected failure

Safety mechanism fails

Fault injected
Building confidence

Structural analysis

Fault analysis

Failure analysis

Least Confidence  Most Confidence

Least Most

\[ \lambda_{\text{Safe}} \]

\[ \lambda_{\text{RF}} \]

\[ \lambda_{\text{MPF,L}} \]
Diagnostic coverage

- DC = % of safety element covered by safety mechanism

\[ \text{Residual} \]
\[ N_{RF} \quad \text{– # residual faults} \]
\[ N_{MPF,L} \quad \text{– # latent faults} \]

\[ \text{Residual} \quad DC_{RF} = 1 - \frac{N_{RF}}{N_{All}} \]
\[ \text{Latent} \quad DC_{MPF,L} = 1 - \frac{N_{MPF,L}}{N_{All}} \]
# A range for diagnostic coverage

<table>
<thead>
<tr>
<th></th>
<th>Structural analysis</th>
<th>Fault analysis</th>
<th>Failure analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unverified</td>
<td>Verified</td>
<td>Unverified</td>
</tr>
<tr>
<td>Safe</td>
<td>286</td>
<td>286</td>
<td>0</td>
</tr>
<tr>
<td>Residual</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dual-point in Safety Function</td>
<td>219</td>
<td>0</td>
<td>215</td>
</tr>
<tr>
<td>Dual-point in Safety Mechanism</td>
<td>2013</td>
<td>0</td>
<td>1704</td>
</tr>
<tr>
<td>Latent</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DC\textsubscript{Residual}</td>
<td>91.0% - 99.7%</td>
<td>91.0% - 99.5%</td>
<td>94.0% - 99.3%</td>
</tr>
<tr>
<td>DC\textsubscript{Latent}</td>
<td>20.3% - 100%</td>
<td>20.8% - 88.2%</td>
<td>26.3% - 87.8%</td>
</tr>
</tbody>
</table>

Continuous refinement
### Example Report

#### All Faults (All Safety Critical Paths)

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Previous Analysis</th>
<th>Current Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unverified</td>
<td>Verified</td>
</tr>
<tr>
<td>Safe faults (outside cone of influence)</td>
<td>0</td>
<td>287</td>
</tr>
<tr>
<td>Safe faults (fault detected by a safety mechanism)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Single-point faults (no safety mechanism)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Residual fault (not covered by safety mechanism)</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Dual point fault (detected/perceived) in safety function</td>
<td>202</td>
<td>0</td>
</tr>
<tr>
<td>Dual point fault (detected/perceived) in the safety mechanism</td>
<td>2035</td>
<td>0</td>
</tr>
<tr>
<td>Dual point fault latent</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td>2245</td>
<td>287</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>2532</td>
<td></td>
</tr>
</tbody>
</table>

#### Number of randomly sampled faults
- **2245 (88.7%)**

#### Design bits
- **2245 (unsafe) / 2532 (all)**

#### Residual Diagnostic Coverage
- **92.10% - 100.00%**

#### Latent Diagnostic Coverage
- **23.97% - 96.68%**
RTL or gates?

- Formal can run on gates, but …
- RTL more likely pessimistic
- Gates likely mask faults

\[
\begin{align*}
DC_{\text{RTL}} &= \frac{N_{\text{RF-RTL}}}{N_{\text{RTL}}} \\
DC_{\text{Gates}} &= \frac{N_{\text{RF-Gates}}}{N_{\text{Gates}}}
\end{align*}
\]

\[N_{\text{RTL}} < N_{\text{Gates}}\]

\[\therefore DC_{\text{Gates}} > DC_{\text{RTL}}\]
RTL to gates equivalency

RTL Structural analysis

<table>
<thead>
<tr>
<th></th>
<th>DC\textsubscript{Residual}</th>
<th>DC\textsubscript{Latent}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>91% - 99.7%</td>
<td>20% - 100%</td>
</tr>
</tbody>
</table>

Gates Structural analysis

<table>
<thead>
<tr>
<th></th>
<th>DC\textsubscript{Residual}</th>
<th>DC\textsubscript{Latent}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>94% - 97%</td>
<td>18% - 98%</td>
</tr>
</tbody>
</table>

• If RTL more pessimistic, gates are unnecessary …
Potential limitations using formal ...

- Large number of formal targets
- Long formal run times
- Large number of inconclusives
- Results biased towards formal friendly designs and design areas
Random sampling

• Confidence interval
  – Allows picking random samples

• Solves
  – Large numbers of formal targets
  – Large numbers of inconclusives
  – Unmanageable results

\[
CI_{99\%} = c \pm \frac{3.38}{\sqrt{n}} \sqrt{1 + 0.59nc(1 - c)}
\]
Goal posting

Intermediate targets

State space

• Possibilities
  – Write temp targets
  – Automatic goal-posting formal engines
  – Seed formal with waveforms
  • Find activity around faults
Automated flow

Safety Definitions

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SR-1</td>
<td>top_module</td>
</tr>
<tr>
<td>1.1</td>
<td>SR-1.1</td>
<td>Permanent Fault leading to wrong results in Register. dist_o &amp; (stb &amp; cyc)</td>
</tr>
<tr>
<td>1.2</td>
<td>SR-1.2</td>
<td>Permanent Fault leading to wrong results in Register. sk &amp; (stb &amp; cyc)</td>
</tr>
<tr>
<td>1.3</td>
<td>SR-1.3</td>
<td>Permanent Fault leading to wrong results in Register. interrupt</td>
</tr>
<tr>
<td>1.4</td>
<td>SR-1.4</td>
<td>Permanent Fault leading to wrong results in Register. ss</td>
</tr>
<tr>
<td>1.5</td>
<td>SR-1.5</td>
<td>Permanent Fault leading to wrong results in Register. sclk</td>
</tr>
<tr>
<td>1.6</td>
<td>SR-1.6</td>
<td>Permanent Fault leading to wrong results in Register. mssi</td>
</tr>
</tbody>
</table>

Structural analysis

```plaintext
module fault_injector(...);
netlist cutpoint -cond ...

dut.top.shift.en
dut.abpl.psel
cover property (...);
... module fault_injector(...);
```

Fault analysis

Automatic properties & constraints!
Handoff to simulation and emulation

Formal Fault Campaign

Fault lists

dut.top.shift.en
dut.abpl.psel
...

Testbenches

module zi_replay_vlog;
  initial begin
    #1;
    force spi_top.wb_rst_i = 1'b1;
    ...

initial begin
  uvm_hdl_force( signal, 1'b0 );
  ...

UCDB and Tracking

Fault injection logic
Automated fault injection for simulation

Fault lists

dut.top.shift.en
dut.abpl.psel
...

Fault injection logic

module fault_injector;
initial begin
    // Read list + pick random fault
    ...
    uvm_hdl_force( signal, 1'b0 );
endmodule
bind testbench fault_injector fi();

Activity Analyzer

Regression simulations

Testbench checks results

vsim +FAULT_LIST=...
+FAILURE_MODE=...
+FAULT_ID=...
...
Fault campaign from the top down

FMEDA
- Fault Analysis (Drive-Works from Elementary level)
- Back-annotate

Tracking
- Safety info
- Fault list
- Reports
- Testbenches
- UCDB

Formal Fault Campaign
- Simulation/Emulation
- Diagnostic coverage
- UCDB
Fault campaign from the bottom up

Safety Definitions

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Description</th>
<th>Safety Path Expression</th>
<th>Primary Safety Mechanism Expression</th>
<th>Secondary Safety Mechanism Expression</th>
<th>Fault Detection Time</th>
<th>Multi-Point Fault Detection</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>top_module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>SR-1-1</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>dir_a &amp; (stb &amp; cyc)</td>
<td>p_error</td>
<td>i_error</td>
<td>5</td>
<td>5 TESTATTRIB</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>SR-1-2</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>ack &amp; (stb &amp; cyc)</td>
<td>p_error</td>
<td>i_error</td>
<td>5</td>
<td>5 TESTATTRIB</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>SR-1-3</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>interrupt</td>
<td>p_error</td>
<td>i_error</td>
<td>5</td>
<td>5 TESTATTRIB</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>SR-1-4</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>sc</td>
<td>p_error</td>
<td>i_error</td>
<td>5</td>
<td>5 TESTATTRIB</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>SR-1-5</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>scilk</td>
<td>p_error</td>
<td>i_error</td>
<td>5</td>
<td>5 TESTATTRIB</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>SR-1-6</td>
<td>Permanent Fault leading to wrong results in Register.</td>
<td>mssi</td>
<td>p_error</td>
<td>i_error</td>
<td>5</td>
<td>5 TESTATTRIB</td>
<td></td>
</tr>
</tbody>
</table>

Tracking

FMEDA

Diagnostic coverage

UCDB

Template + back-annotate
Summary

• Formal provides …
  – Quick and easy fault categorization for worst-best case DC
  – No environment setup required – no testcases
  – High-level of confidence in results – can’t beat a proof!
  – Ties in with simulation and emulation
  – A great front-end for the entire fault campaign process
Requirement Tracing in the ISO26262 World

Charles Battikha (chuck_battikha@mentor.com)
Why Requirements matter…

- Example: NASA’s Mars Climate Orbiter
  - Sent crashing into Mars by NASA
  - The Orbiter spoke to NASA in metric… But the engineers on the ground were replying in non-metric English

“What is being designed, built, and verified is based on requirements and thus intended”
Safety Requirements

Safety Goal met? = Complete List + Each True

Safety Goals -> Functional Safety Requirements (FSR) -> Functional Safety Concept

Allocation -> Item Architecture

Technical Safety Requirements (TSR) -> Technical Safety Concept

Allocation -> Chip & Software Architecture

Software Safety Requirements

Hardware Safety Requirements

26262 6-6
26262 5-6

Achieving ASIL rating means meeting all requirements
Why are requirements hard to write?

• Human Language is inherently vague and imprecise

• Relying on engineer’s writing skills….  
  – Desire to jump into the details…

• Trouble separating WHATs from HOWs  
  – Good enough…  
  – Let’s get on with it…

• Believe spending time writing requirement will cause delays  
  – Good enough…  
  – Let’s get on with it…
Requirements: Common Problems

• Errors of Omission
  – What was intended, was not actually stated; Important information left out

• Errors of Commission
  – Information is wrong; Information is contradictory

• Errors of Clarity
  – Requirements stated in ways that lead to confusion, misunderstanding
  – Creation of assumptions

• Errors of Understanding
  – Ambiguous, words get in the way
  – Each person internalizes and applies their own definitions
Writing Safety Requirements

- Natural language
- Informal notation
- Semi-formal notation (syntax defined)
- Formal notations (syntax & semantics defined)

ASIL A/B

ASIL C/D

Increase Effort
Increase Rigor
Reduced Risk
Defining Requirements

• Define WHAT not HOW
• Should be:
  – Complete / Atomic
  – Consistent
  – Comprehensible
  – Realistic / Feasible
  – Verifiable
  – Valid / Correct
  – Necessary
• Each requirement should be written with a standard style and contain the following components:
  – **Action**: Operation design will perform. Atomic and unambiguous.
  – **Condition**: Under what conditions is the action performed.
  – **Testable Result**: What will occur. Should be specific.
  – **Reaction Time**: A bounding time. For instance, a ‘within’ time frame.

• Time should in the proper context. Stay away from implementation details.
Creating Safety Requirements

Sense
Define Command
Define Information
Detection Requirement

Compute
Define equations for faults to transition to Safe State
Reaction Requirement

Act
Define Safe State

Requirements should be testable - can be viewed as preliminary test cases.
Value of Tracing

Top Down Trace – Find **unallocated** and/or **unimplemented** requirements

- System Requirements
- Hardware Design Requirements
- Hardware Design Documentation
- Hardware Design Implementation

Bottom Up Trace – Find **unnecessary**, **unneeded**, **unwanted** functions or features

Top Down Trace – Find **untested** requirements, **failing** requirements, **unimplemented** tests

- System Requirements
- Hardware Design Requirements
- Verification Plan
- Test Bench Implementation
- Testing Artifacts

Bottom Up Trace – Find **undocumented** testing
Testing Artifacts

• Requirements must trace into the testing artifacts
  – Shown to have been actively tested and shown to pass
  – For simulations, typically UCDB and/or Test Log Files
  – Artifacts from a “Run for the Record” regressions used for final reports.
Directed Testing & Requirements

- Directed Tests are often used for 1-1 match of requirement to test
- However, typical Directed Tests driven to satisfy requirements tend to have shortcomings:
  - Not complete in testing across the full design
  - Down stream errors are not checked
  - Are limited to specific times, situations
Random Testing & Requirements

• Random Testing and UVM Test Benches allow a smaller set of Test Cases to address multiple requirements concurrently.

• Random testing of requirements & checking for passing is the AND of:
  – Test Case Passing
  – Appropriate Stimulus Generated
  – Appropriate Prediction Generated
  – Results are checked and match

• Checks distributed work across test cases, predictors, and scoreboards
  – AND function can be addressed by Functional Coverage
Tracking in a Test Bench

• Logging for traceability occurs where testing of a requirement is done
• Typically is an ‘else’ in an error check
• Simple Functional Coverage is okay IF run for record must achieve 100% passing test cases

if (expected_crc != actual_crc)
    `uvm_error("DUT generated bad CRC")
becomes
// [Implements: VREQ_nnn]
if (expected_crc != actual_crc)
    `uvm_error("DUT generated bad CRC")
else begin
    `uvm_info("DUT generated good CRC")
    Add to specific VREQ to covergroup
end

[DES_REQ_nnn] When sending a message out on the channel, the design shall calculate a CRC in accordance with ...

[VREQ_nnn] The Test Bench shall have a checker on DUT channel output that ensures all messages generated by the design have a correct CRC....
Tracking in a Test Bench

• If failing test cases can occur in run for the record,
  – Passing test cases may have set functional coverage
  – Create passing / failing covergroups
  – Coverage of failing conditions trumps good covergroup.
  – Parsing log files can accomplish similar tracking

```plaintext
// [Implements: VREQ_nnn]
if (expected_crc != actual_crc) begin
  `uvm_error("DUT generated bad CRC")
  Add specific VREQ to bad covergroup
end else begin
  `uvm_info("DUT generated good CRC")
  Add specific VREQ to good covergroup
end
```
Tracking in a Test Bench

• UVM Test Benches distribute work so checking may be too simplistic for tracing
  – Scoreboards may simply compare expected data against actual data
  – May not be possible to isolate checks to a specific requirement
• Usually the ‘predictor’ can be associated with a requirement
  – A requirement would then be considered passing if:
    • The Predictor made appropriate prediction
    • Test Case has passed (no scoreboard miscompares)
  – Coverpoints created to AND these conditions
Predictor to Requirement Mapping

• For some designs, it may be possible to create a more direct predictor/checker mapping to requirements
  – Tradeoff of complexity in checkers versus complexity in tracking
Assertions -> Requirements

```
// formal randomly picks a bit(s) to flip.
asm_mask: assert property ( $countones(one_error_mask) == 1 );
// Check ECC repair is correct
req_nnnn: assert property ( fixed_data[7:0] == data );

-- XOR mark to flip 1 bit
one_error_data <= one_error_mask XOR encoded_data;
fixed_data <= ecc_correction_function(one_error_data);
```

```
function [12:0] ecc_calc( data, ... );
  wire logic p1 = 1 ^ data[0] ^ data[1] ^ ...
  wire logic p2 = ...
  ...
  return ({data[7],data[6],data[5],data[4],p8,data[3], ... } );
endfunction

// Check ECC calculation
req_xyz: assert property ( encoded_data == ecc_calc(data, ... ));
```

Assertions can also be assigned per requirement.

A proved assertion is positive coverage.
Requirements Management

After release to production. Changes may trigger for potential re-certification.
Requirements Tracing Tools

- A centralized view that connects the development process and results
- Traceability at all stages of development
- Quickly understand the impact of a change across the project
- Reflects the current status of the project using live data
Summary

• ISO26262 defines:
  – Top down flow of safety requirements
  – Requires precise language for requirement definition
  – Traceability
  – Change Management and Source Control

• Poor requirements creates an unstable base to build on

• Tracing should be done into verification artifacts
Questions?
Contact Information

Charles Battikha (chuck_battikha@mentor.com)
Doug Smith (doug_smith@mentor.com)

https://www.mentor.com/mentor-automotive

White paper - “How Formal Reduces Fault Analysis for ISO 26262”
http://go.mentor.com/4QQrY