
# DVClub Europe

## Formal fault analysis for ISO 26262 fault metrics on real world designs

Jörg Große

Product Manager Functional Safety

November 2016

# Introduction Functional Safety

The objective of functional safety is freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly.

| Functional Safety Risks | Risk drivers | Risk management through functional safety standards |
|---|---|---|
| • Systematic Failures<br>  ○ Design errors<br>  ○ Tool errors<br>• Random failures<br>  ○ Hard errors<br>  ○ Soft errors | • Continuous increase in flow and tool complexity<br>• Continuous increase in functionality<br>• Increasing density of the design process node<br>• Decreasing energy levels | • Minimize systematic errors<br>• Safeguard against random errors |

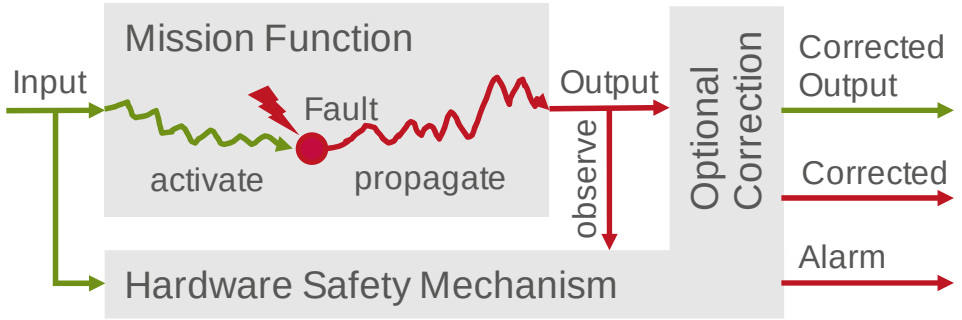
# Safeguard Against Random Errors

Fault lifecycle:

- Activated (reached)
- Propagated to mission output
- Observed by safety mechanism
- Detected/Corrected by safety mechanism

Faults are caused by hard or soft errors

- Permanent (Latch up, bridging)
  - ➢ StuckAt's
- Transient
  - ➢ Single Event Faults
- Intermittent

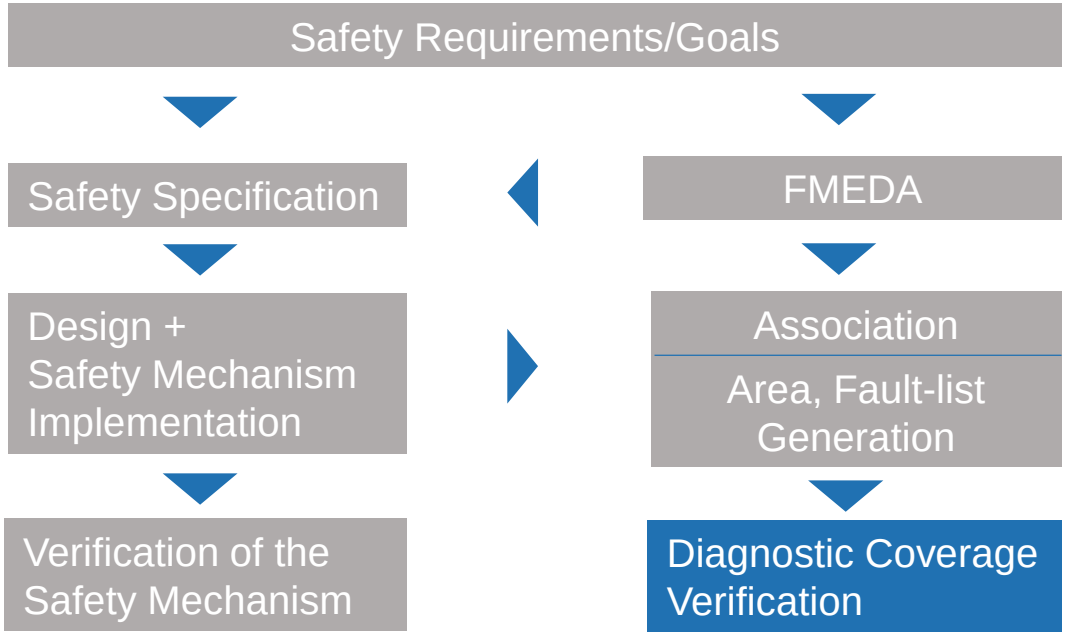



ISO 26262 requires

- Verification of the safety mechanism
- Diagnostic coverage of the safety mechanism

# Safety Flow for Random Errors



# Single Point Fault Metric

## ISO 26262 provides and requires the framework

### 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”

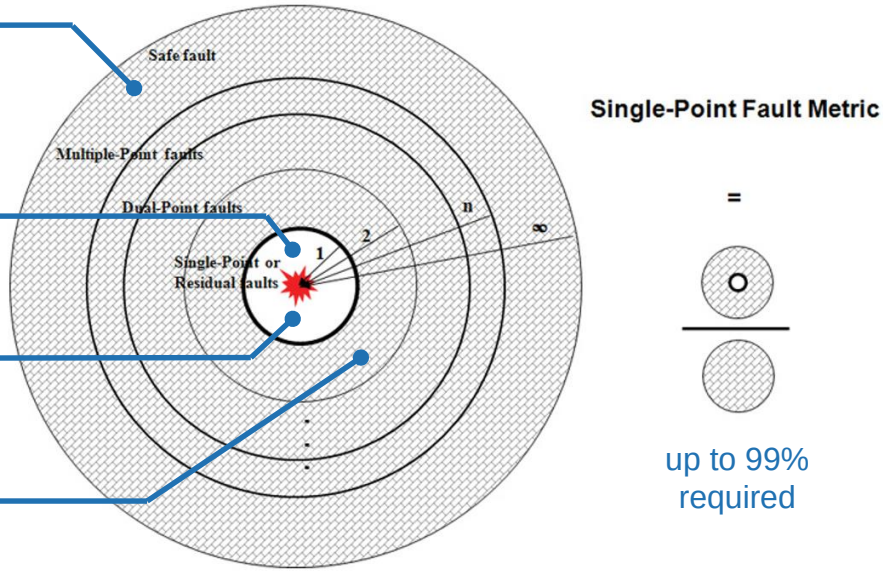


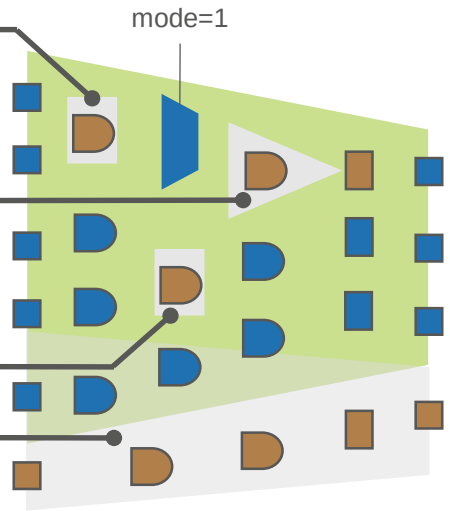

Diagram Courtesy International Standards Organization (ISO)

# Reasons for Safe Faults

## Considering StuckAt faults only

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




**All safe faults cannot propagate to observation points**

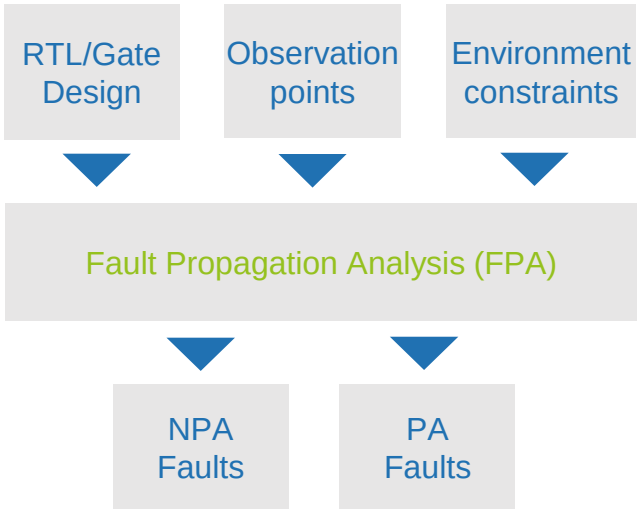
- Mission outputs
- Internal registers

# Formal Fault Propagation Analysis

## No formal knowledge required

- Automatically classifies faults into
  - Non-propagatable faults (safe)
  - Propagatable faults
    - Dangerous (single point & residual)
    - Potentially detected (multipoint detected)
- Can be performed at RT and gate level
- Only needs very limited additional input
  - Observation points define where faults can propagate (default is primary outputs)
  - Environment constraints, assign values to test pins, debug modes and control registers
- Fault lists can be provided by the user or generated by the tool
  - StuckAt fault model supported

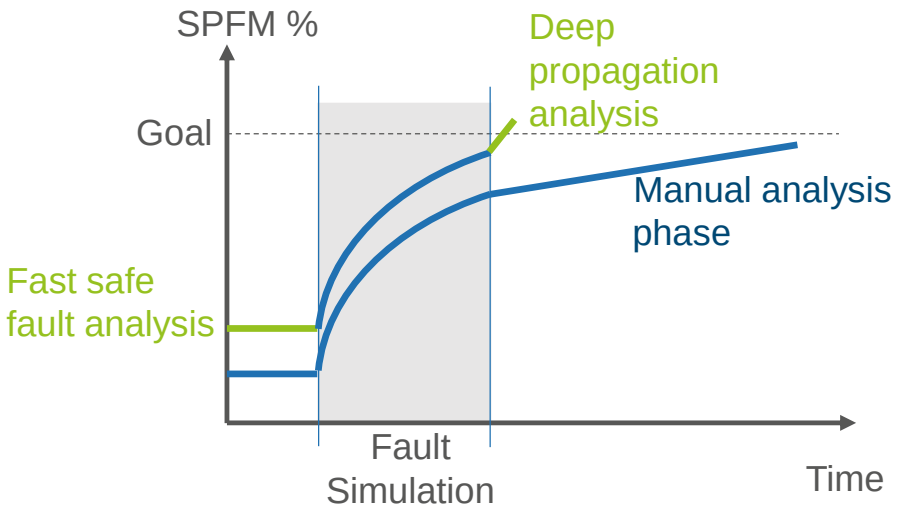


# Use Case: Formally Identify Safe Faults

## The more the better

- Today engineers use fault simulation to determine fault metrics
- What if the fault simulation does not achieve the desired metric?
  - Manually identify safe faults
  - Improve safety mechanism, re-simulate …
  - How many safe faults are there?
- Formal fault propagation identifies safe faults automatically
  - Fast up front analysis to increase the safe fault population
  - Deep propagation analysis to close the remaining gap
  - Reduce time and increase confidence




Only formal technology can efficiently combine the design and environmental constraints to bring fault analysis to the next level.

# Fast Fault Propagation Analysis

- Always executed for all possible fault locations
- Shall be executed before fault simulation and/or deep fault propagation analysis

| | RTL example | Gate level |
|---|---|---|
| Description | open core processor<br>~10k lines / 794 registers<br>12.260 faults | summary of field data<br>up to 2 million faults |
| Performance – CPU time | 2 minutes | up to several hours |
| Result | 3257 (26%) safe faults | up to 14% safe fault<br>(including untestable) |

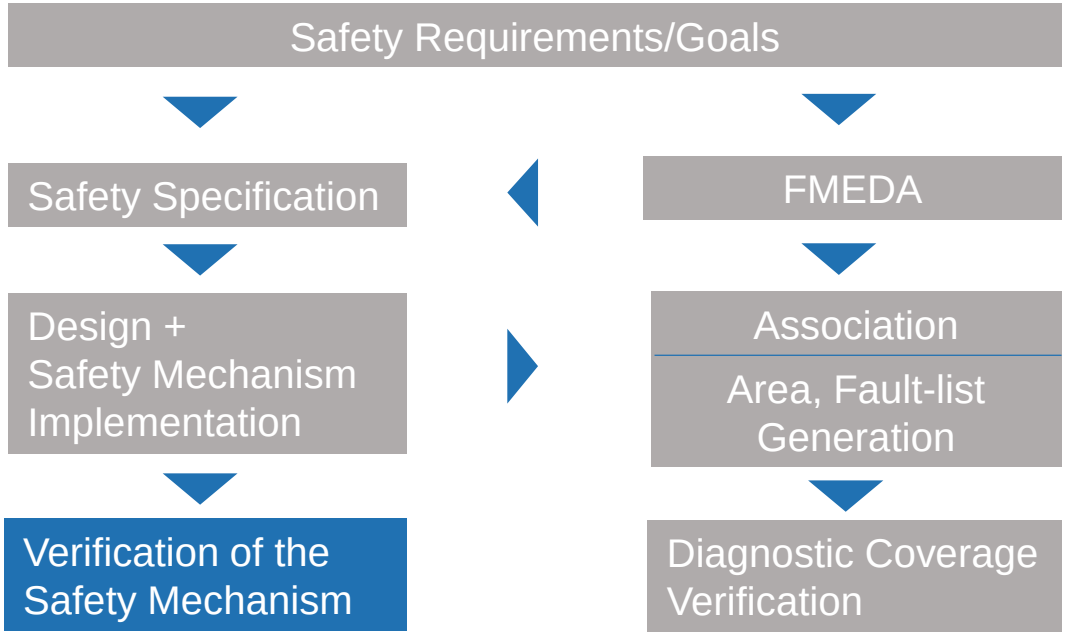
# Deep Fault Propagation Analysis

- Requires a fault list as input
- Often executed in fault sampling mode or on smaller sets of faults

| | RTL example | Gate level |
|---|---|---|
| Description | open core processor<br>~10k lines / 794 registers<br>12.260 faults | summary of field data<br>up to 2 million faults |
| Performance – CPU time | most safe faults typically proven in 20-30 sec<br>(some faults cannot be proven within time budget) | several seconds to minutes per fault<br>(some faults cannot be proven within time budget) |
| Result | ~13% safe faults<br>(after fast fault analysis, NOTE: design not “optimized” by synthesis) | 0.3-2.1% safe faults<br>(after fast analysis or fault simulation) |

# Safety Flow for Random Errors



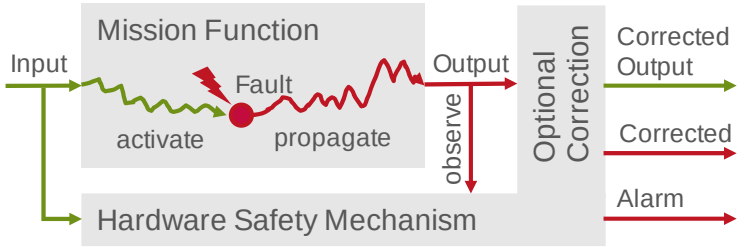

  

# Verification of the Hardware Safety Mechanism

## ABV with fault injection

### Problem

- Hardware safety mechanism is inactive unless a fault occurs
  - ISO 26262 recommends fault injection for verification
- Fault Injection complexity for bit vectors:
  - $2^{width}$ possible data input combinations
  - *(width)* 1-bit errors
  - *(width* width-1)* 2-bit errors
- Simulation based verification hits its limits
  - Must anticipate all possible input combination
  - Propagate all possible faults
  - Check the outcome



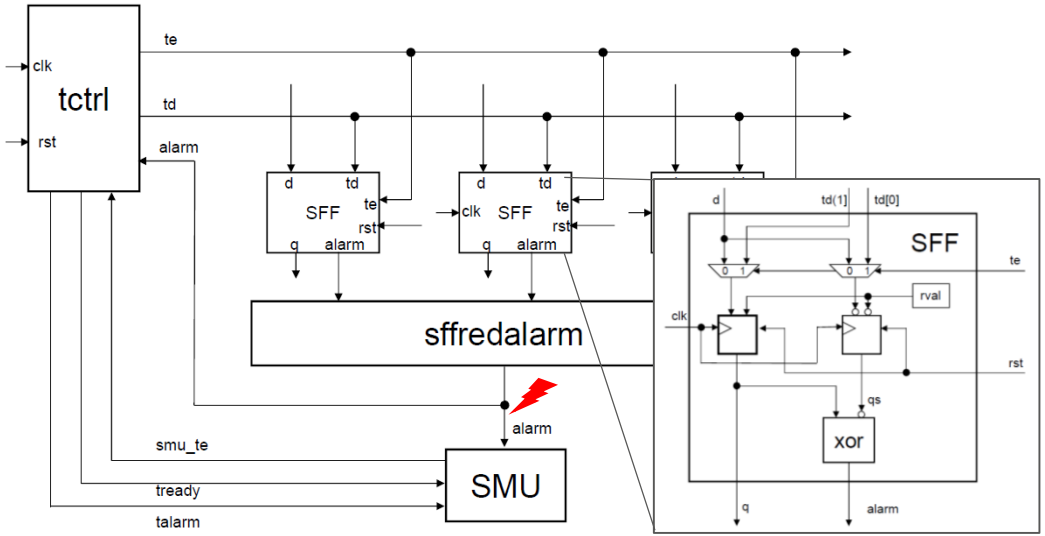

### Solution

- Formal verification with fault injection
- Trivial output assertion
  - No “Alarm” when no fault is injected
  - Always expect “Alarm” when fault is injected
- Inputs must be constrained
  - Avoid illegal input pattern when no fault is injected and no alarm is expected

# Formal Safety Verification with Fault Injection

## Where the rubber meets the road




H. Busch, Infineon Technologies, “Formal Safety Verification of Automotive Microcontroller Parts”, ZuE2012, Bremen

### OneSpin 360™ DV-Verify with fault injection

- Exhaustive fault activation including multiples
- Efficient modeling of faults and specification of requirements
- Highly automated
- Handling large blocks and huge number of faults simultaneously

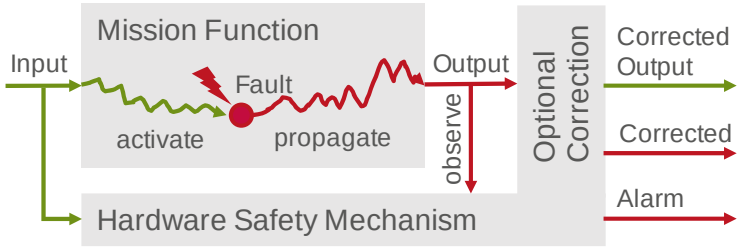
  

# Safety Critical Verification

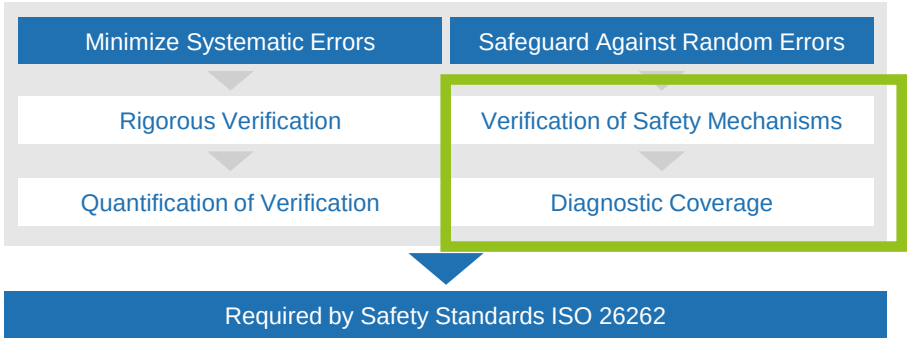
## Assuring High Reliability

Meeting tough functional safety standards, e.g. ISO 26262

Minimize systematic errors & protect against random faults




- Rigorous, exhaustive formal
- Industry-leading quantification and qualification of formal properties
- Efficient verification of functional safety mechanism
- Precise diagnostic coverage through formal fault analysis




# Thank You!