Next Generation ISO 26262-based Design Reliability Flows

Jörg Große  &  Sanjay Pillay

www.onespin.com  www.austemperdesign.com
Powerful Solutions Require Strong Technology and Apps

**Advanced Design Verification**
- Agile Design Evaluation
- Metric-Driven Verification
- Block Integ. Validation

**Innovative Specialized Solutions**
- FPGA Impl. Verification
- Safety Critical Verification
- C++/SysC Des. Verification

- Automated Inspection
- Design Exploration
- Sequential EC/RTL
- Assertion Verification
- Operational Assertions
- Observation Coverage
- DV Apps
- X-Prop
- Connect
- Register
- Scoreboard
- Protocol Comp. Activation
- SystemC/C++ Arithmetic
- Sequential EC FPGA
- Fault Propagation
- Fault Detection
- Fault Injection
- Security Verification
- Specification Verification

**Proof Engines**
- SystemVerilog
- VHDL
- SystemC
- SVA
- PSL

**Formal Model**

**Adv. Debug**

**LaunchPad**

---

**Jörg Grosse – Product Manager Functional Safety**
• Provider of **End-to-End Functional Safety** Tool suite for Automotive, Industrial, Medical and Enterprise Markets.

• One-stop solution for ASIC vendors to **analyze, augment** and **verify** their designs for Functions Safety Compliance

• Based in Austin, TX, USA; Founded 2015

• Tools in production with customers

Sanjay Pillay  
Founder & CEO

Previously responsible for:

- World wide enterprise SSD controller SoC development at HGST/STEC
- World wide SoC development at TRIDENT/NXP/CONEXANT
- Head of audio development at MAXIM
- Functional safety consultant
Agenda

- Introduction
- The design - AXI Crossbar
  - **Austemper** - Insert safety mechanism – STEP 1
  - **OneSpin** - prove that insertion did not corrupt main functionality
  - **OneSpin** - identify faults missed by safety mechanism
  - **Austemper** - insert additional safety mechanisms – STEP 2, STEP 3
  - **OneSpin** - prove that insertion did not corrupt main functionality
  - **OneSpin** - prove that safety has improved
  - **OneSpin** - identify/debug undetected faults
- Integration with Fault Simulation
- Results and Conclusion
  - ✔ Real design
  - ✔ Hands-on tutorial
  - ✔ Questions welcome
Functional Safety and Safety Mechanisms

Objective:
Freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly

Functional Safety Risks

• Systematic Failures
  o Design errors
  o Tool errors
• Random failures
  o Hard errors
  o 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

Safety mechanisms prevent/control random hardware failures
Types of Safety Mechanisms

SOFTWARE MECHANISMS
- Self-Test Routines
- Watchdog Timers

HARDWARE MECHANISMS
- Error Correcting Codes
- Parity bit
- Lockstep
- TRM with Voting Logic
- LBIST
Fault Classification and Metrics

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"

Safe Faults: do not propagate to outputs
Detected Faults: propagate to outputs but detected by safety mechanisms
Dangerous Faults: propagate to outputs and missed by safety mechanisms

Diagram: Courtesy International Standards Organization (ISO)
Observation and Diagnostic Points

Note: faults propagating to observation points but not to diagnostic points are definitely dangerous
The Candidate Design

AMBA AXI Fabric
- 2 Master ports
- 2 Slave ports
- Separate Read and Write channel FIFOs
- Configurable FIFO depth
- Single Clock Domain

Functional Safety
- None

Data FIFO
# Austemper Safety Synthesis

## FEATURES

<table>
<thead>
<tr>
<th>FEATURES</th>
<th>Annealer</th>
<th>RadioScope</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ERROR DETECTION &amp; CORRECTION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamming code based $n$-bit detect/$m$-bit correct</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Structures supported</td>
<td>RAM, ROM, Reg Files, FIFOS, stacks</td>
<td>Flip-Flop Banks</td>
</tr>
<tr>
<td>User –Defined Structure selection</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Auto-Grouping of Structures</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td>User selectable Option (Parity vs EDC vs ECC)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multi-pass w/ incremental safety insertion mode</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>FAULT TOLERANCE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redundancy</td>
<td>Macro/Module level</td>
<td>Localized Logic cones</td>
</tr>
<tr>
<td>Duplication/Triplication</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multi clock designs</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Auto-Identification</td>
<td>Memories</td>
<td>State Machines</td>
</tr>
<tr>
<td><strong>PROTOCOL CHECKS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Covered Items</td>
<td>Interface Parity/protocol, FIFO overflow/underrun</td>
<td>FSM Valid states &amp; transitions</td>
</tr>
</tbody>
</table>
Safety Synthesis Steps

**STEP 1**
- Use RadioScope to insert parity protection on selected state elements

**STEP 2**
- Use RadioScope to insert end-to-end datapath parity

**STEP 3**
- Use Annealer to duplicate register blocks
STEP 1: RadioScope

TOOL INPUTS

DESIGN FILES: Verilog
DESIGN TYPE: RTL
SYNTHESIS FILE: List of Target State Elements

TOOL OUTPUTS

DESIGN FILES: with parity inserted and built-in safety alarms
DESIGN TYPE: Verilog RTL
ERROR CHECK: Verilog Test bench and Test cases.
EQUIVALENCE CHECK: script to verify absence of corruption with third party tool
DEMO AUSTEMPER
STEP 1 Output

axi_cross_wrapper_ATD

axi_cross

Control/Observe Parity

Safety Err Out

Austemper Additions

Master0 RD Port

Master1 RD Port

RD_CHANNEL

Data FIFO

PGC

Slave0 RD Port

Slave1 RD Port

Slave0 WR Port

Slave1 WR Port

Master0 WR Port

Master1 WR Port

WR_CHANNEL

Data FIFO

PGC

PGC

PGC

PGC

PGC

PGC

PGC
Verify Safety Mechanism

- Original design functionality corrupted?
  - Use Combinational/Sequential Equivalence Checking
Verify Safety Mechanism

- Safety Mechanism detects **enough** faults?
  - Verify diagnostic coverage
Formal Fault Analysis Flow

1. Safety-Critical Function
2. Hardware Safety Mechanism
3. Input
4. Observation Points
5. Diagnostic Points
6. Fault List
7. Input Constraints
8. Fault Classification Report/DB
DEMO ONESPIN
Results

• OneSpin 360 EC: we have proven that functionality has not been corrupted
• OneSpin 360 DV: low fault coverage

➢ Additional safety mechanism might be required
STEP 2: RadioScope

TOOL INPUTS

DESIGN FILES: Step 1 Verilog
DESIGN TYPE: RTL
SYNTHESIS FILE: List of Datapath Signals

TOOL OUTPUTS

DESIGN FILES: Verilog with E2E parity inserted and built-in safety alarms
DESIGN TYPE: RTL
ERROR CHECK: Verilog Test bench and Test cases.
EQUIVALENCE CHECK: script to verify absence of corruption with third party tool
DEMO AUSTEMPER
STEP 2 Output

Austemper Additions
STEP 3: Annealer

**TOOL INPUTS**

- DESIGN FILES: Step 2 Verilog
- DESIGN TYPE: RTL
- SYNTHESIS FILE: List of Macroblocks to duplicate

**TOOL OUTPUTS**

- DESIGN FILES: Verilog with duplication and Checkers with built-in alarms
- DESIGN TYPE: RTL
- ERROR CHECK: Verilog Test bench and Test cases.
- EQUIVALENCE CHECK: script to verify absence of corruption with third party tool
Verify Safety Mechanisms

- Original design functionality corrupted?
  - Use Combinational/Sequential Equivalence Checking
• Safety Mechanism detects **enough** faults?
  – Verify diagnostic coverage
DEMO ONESPIN
Results

• OneSpin 360 EC: we have proven that functionality has not been corrupted
• OneSpin 360 DV: additional safety mechanisms detect previously undetected faults
• OneSpin 360 DV: identify/debug dangerous faults
Integrating Formal with Fault Simulation

• Analysis of software safety mechanisms requires fault simulation
  – Formal tools cannot read self-test software routines

• Analysis of large SoCs requires fault simulation
  – Formal tools have capacity limitations

• Can formal verification still help in these circumstances?
  – Yes!
Integration of Formal FPA with Simulation

- Two-mode approach fits well with simulation flow
KaleidoScope: Austemper Fault Simulator

- Safety Context derived entirely from RTL simulations
- Concurrent fault propagation without restrictions
- ~4 orders of magnitude faster than GLS
- Auto-classification of Fault outcomes
- Integration with Analysis front-end for computing DC
- Smart Fault injector automates the process
- Unresolved faults dispatched via either

- Hybrid Simulation
- Formal – Deep Analysis
KaleidoScope: Austemper Fault Simulator
Conclusions

• Hardware safety mechanisms detect random hardware faults

• Hardware safety mechanisms must be verified
  – Do not corrupt normal functionality
  – Detect enough faults, depending on target SIL

• Austemper tools automatically insert a variety of safety mechanisms
• OneSpin Safety-Critical Solution automates verification tasks

• Efficient and streamlined flow to ISO 26262 Certification
References


2. S. Marchese - J. Grosse – Formal fault propagation analysis that scales to modern automotive SoCs, DVCon Europe 2017

3. S. Marchese - Using formal to verify safety-critical hardware for ISO 26262, OneSpin Solutions White Paper

A note to offline readers: to receive a video of the demo parts of this tutorial please contact

Joerg.Grosse@onespin.com
Sanjay.Pillay@austemperdesign.com
Questions?