Exhaustively Verify SEU Mitigation Techniques Using Formal Verification

Jeremy Levitt
Questa Formal R&D

June 27, 2017
Today’s Agenda

• Introduction & Scope

• The Problem: Verifying SEUs With Simulation

• The Solution: Automated Formal Analysis

• Case Study
Introduction: Scope

- This talk focuses exclusively on digital circuitry specified in a register transfer level (RTL) language (VHDL or Verilog)
  - “Safety Mechanisms” to detect and correct/flag faults
    - TMR, ECC, onehot FSM encoding, Etc.
  - Faults
    - SEU, Stuck-at-one/zero, Transient, Etc.

- We do not go into any analog, physical, or “side-channel” effects

- We do not address higher level software
The Problem: Verifying SEUs with Simulation

- Simulation is only as good as the test written — garbage in/garbage out

- Overhead: must make required safety mechanisms have been added before you can even start testing mitigation logic

- Parsing & sorting results with scripts is time consuming and error prone

- Writing tests to force values at memory elements and check for corrected results also time consuming and error prone

**Bottom line:**
SEU phenomena can not be exhaustively verified by simulation-based approaches
Another approach: Formal Verification

Use formal to mathematically prove outputs $A = B$ for the same inputs, for all time

$A = \text{Design} + \text{TMR}$

$B = \text{Design} + \text{TMR} + \text{SEU}$
The Key to Advantage to Fault Verification w/Formal: Formal Proofs Are Valid for All Time

Analogy

Finding solutions to $ax^2 + bx + c = 0$

- Constrained-random simulation approach: randomly plug-in numbers in the hope you eventually satisfy the equation

- Formal approach: algebraically compute the solutions (to the Boolean equation representing the DUT)

✔ With formal, exactly when SEU event happens doesn’t matter – if the inserted SEUs affect the desired output the proof will “fail”
An Automated Approach: Questa Sequential Logic Eq Checking (SLEC) App

What: Formally verify A & B do the same thing even if different # of states

Why: Exhaustive verification orders of magnitude faster than simulation
Questa Fault Injection and Equivalence Flow
Targeted SEU and stuck-at fault analysis without a testbench

Formal-based flow focused on validating the Safety Mechanisms' successful detection and handling of faults

**Inputs:**
A = DUT logic + “Safety Mechanism”
B = A + faults injected

**Desired output:**
Does the output of the fault case match the normal case?
i.e. does Safety Mechanism detect the fault, and react properly?

**Benefit:**
Exhaustively prove which faults affect functional safety

Done!

Failure Rates Met?
Yes

No → Improve Safety Mechanisms

DUT + Safety Mechanism
Fault Generation and Reduction
DUT + injected faults + Safety Mechanism

Questa SLEC

Benefit:
Exhaustively prove which faults affect functional safety
Questa Fault Analysis: Fault Pruning

Reducing the set of faults that need to be fault injected

- **A subset of faults**
  - Only a subset of faults in a given design will affect the safety requirement. They are in the COIs of the safety critical signals.

- **Safe elements**
  - Design elements not in the COI of a safety critical signal are automatically considered safe.

- **Configurations and constraints**
  - The COI can be reduced further by applying top-level constraints such as disabling DFT, debug and test, or other non-operational modes.
Questa Fault Analysis: Safety Mechanism

Safety Mechanisms reduce fault Injection requirements

- Detectable fault
  - Design elements in the COI of a safety requirement, and
  - *Overlap* with the COI of the associated safety mechanism

- Undetectable fault
  - Design elements in the COI of a safety requirement, and
  - *Not* in the COI of the safety mechanism
  - Must be considered a dangerous fault

Injected faults here are **undetectable** if no COI overlap
(Undetected Dangerous Fault)

Hardening is to drive more overlap
**Questa Fault Analysis: Bottom-Up Methodology**

*Perform early at the block-level and aggregate to the top-level*

- Safety mechanism can be designed-in with critical functionality
- Fault pruning can be done early to understand Diagnostic Coverage

---

**Diagram Description: System Interconnect**

- **AXI Bus** connects to the AXI Bridge, which is connected to the Ethernet Controller through the PHY interface.
- The SDRAM Controller connects to the SDRAM through the AXI Bridge.
- The DMA interfaces are connected to the System Interconnect 0 and System Interconnect 1.
- The μC (Microcontroller) is connected to the System Interconnect 1.
- The Graphics Engine is also connected to the System Interconnect 1.

**Legend:**
- **Hard IP** (gray)
- **Soft IP** (blue)

---

Restricted © 2017 Mentor Graphics Corporation
Example Results

Only the fault path output is not equivalent
Questa Fault Injection and Equivalence

Dangerous Undetected (DU) Fault

- Injected fault
- Output detected
- Undetected by checker
Formal Fault Coverage

<table>
<thead>
<tr>
<th>Sec #</th>
<th>Section / Coverage Link</th>
<th>Type</th>
<th>Goal</th>
<th>Coverage</th>
<th>% of Goal</th>
<th>Status</th>
<th>Weight</th>
<th>Link Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>SPI Formal Fault Verification seu</td>
<td>Testplan</td>
<td>100%</td>
<td>84.21%</td>
<td>84.21%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.1</td>
<td>correctness</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.2</td>
<td>detectable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3</td>
<td>maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>84.21%</td>
<td>84.21%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Fault 1 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Fault 2 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Fault 3 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Fault 4 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Fault 5 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Fault 6 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Fault 7 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Fault 8 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.9</td>
<td>Fault 9 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.10</td>
<td>Fault 10 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.11</td>
<td>Fault 11 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.12</td>
<td>Fault 12 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.13</td>
<td>Fault 13 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.14</td>
<td>Fault 14 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.15</td>
<td>Fault 15 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.16</td>
<td>Fault 16 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.18</td>
<td>Fault 18 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.19</td>
<td>Fault 19 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
<tr>
<td>3.3.20</td>
<td>Fault 20 maskable</td>
<td>Testplan</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
<td>Clean</td>
<td>1</td>
<td>Clean</td>
</tr>
</tbody>
</table>
Early Adopter Success at EU Auto Systems Supplier

- **Verification Project Outline**
  - Exhaustively verify fault tolerance of ABS digital logic
  - Small team, new to formal: an automated approach is essential

- **Partnership Setup**
  - Close methodology partnership with MGC and customer engineers
  - Built tailored, automated solution on top of Questa SLEC app

- **Success to date**
  - 2000 fault points (critical registers)
    - 10% are shown to be unsafe, 90% **proven** safe
    - Total run time under 1 hours
“Our products can’t compromise on safety, so our verification must be to 100% complete.

As such, formal-based verification solutions are essential given the exhaustive nature of the analysis. For some IP Blocks formal is the only option to perform a complete verification.

With the Questa SLEC formal app it is possible to quantify the fault coverage of certain IP Blocks e.g. safety critical IP Blocks.”

Thorsten Ehrenberg
Manager, Safety Microcontroller Development
Continental Teves Chassis & Safety Division
Fault Proof: Using Formal Techniques for Safety Verification and Fault Analysis

Adrian Traskov, Thorsten Ehrenberg, Sacha Loitz
Continental Tires AG & Co. eHG, Frankfurt a. M., Germany
Abdoulaah Ayati, Avidan Efoy, Joseph Humphrey III

Mentor Graphics

Abstract—Safety mechanisms designed to correctly detect random hardware failures implement critical functionality but are relatively rare in gate count, which often makes them an ideal application for formal verification techniques. In this paper, we present a case study describing fault analysis of a Triple Modular Redundancy (TMR) element and its associated majority voter using formal. We start by describing a generic flow for fault analysis of safety mechanisms, including fault population reduction, fault injection, checking and classification, and collection of metrics. We then move on to show how formal can be used to perform each of these tasks in the context of a TMR safety mechanism. Finally, we compare formal results and run time against those obtained using dynamic simulation techniques, and show how formal is able to minimize the analysis effort required.

Keywords—ISO 26262, fault analysis, formal verification, property checking, single event upset, formal verification of safety mechanism

I. INTRODUCTION

One of the cornerstones of ISO26262-compliant design is the inclusion of additional logic to implement a “safety mechanism” whose purpose is to ultimately guarantee safe behavior of the system against the fail of

DVCon Europe, October 2016

10.2, Fault Proof: Using Formal Techniques for Safety Verification and Fault Analysis

http://events.dvcon.org/events/browseproceedings.aspx?confid=211
Summary

- Verifying safety and flight-critical systems’ vulnerability to transient (and persistent) logic faults is ethically and legally mandatory.

- Simulation-based approaches are not exhaustive, creating some risk of bug / fault escapes.

- Exhaustive SEU verification with an automated, formal-based analysis greatly increases the quality of results and reduces risk.