Formal Verification for Safety-Critical Applications of FPGAs

David Landoll, Solutions Architect, OneSpin Solutions
Agenda

• Three industry trends
  • Growth in the type of applications that must meet safety standards
  • Replacement of ASICs and custom chips by FPGAs
  • Increasing use of growth formal technologies for verification

• Three safety applications for formal verification
  • Elimination of designed systematic errors
  • Elimination of introduced systematic errors
  • Reduction of random error effects

• Three real-world examples
The Need for Safety

- Historically only certain types of designs were safety-critical
  - Military/aerospace applications
  - Implanted medical devices
  - Nuclear power plants (of course!)

- Safety is becoming an issue for many more types of products
  - Autonomous vehicles (of course!)
  - Security systems
    - No one wants to be locked out or have their alarms disabled
  - Cell phones
    - If no landline, cell phones are the only way to summon help
Safety Standards

• There are many standards related to safety-critical designs
  • IEC 61508, IEC 61513, IEC 61511, IEC 62279, IEC 6206, DO-254, ARP4754, ISO 26262, MISRA…

• Common themes from the standpoint of electronic (chip) design
  • Must eliminate systematic errors (bugs) from the design itself
  • Must mitigate effects of random errors such as alpha particle hits
  • Must provide evidence (high coverage metrics) for both steps
  • Must have well organized and documented processes
    • Applies to both design and verification

• Customers in the supply chain may also demand certification of standard compliance from a third-party agency
The Role of FPGAs

• Historically, FPGAs were used for small designs with low volume
  • Very limited support for processors, engines, complex I/O, etc.
  • Development time and cost for ASIC could not be justified
  • Designs were debugged in the bring-up lab
    • Devices could be reprogrammed to fix bugs
    • “Pre-silicon” verification was minimal
• Today’s FPGA is a full-fledged system-on-chip (SoC)
  • Commonly replacing ASICs and some custom chips
  • No longer possible to bring up huge designs in the lab
    • Detect-debug-fix-reprogram-verify cycle takes days
  • Many FPGA projects adopting ASIC verification flows
The Role of Formal Verification

- Formal techniques have become mainstream in recent years
  - Very low chance of simulation hitting every corner-case bug
  - Formal analysis is only exhaustive method for verification
  - “Apps” and improved automation make formal much easier to use
- Formal can meet the tough requirements imposed by safety standards

Simulation relies on hand-written stimulus to expose states to be checked

Formal applies checks to all states and transitions, for exhaustive testing

Error-prone, time consuming

Exhaustive test, no stimulus effort
Combination of the Trends

More types of safety-critical designs

Replacement of ASICs with FPGAs

Expanded usage of formal verification

Increasing reliance on formal verification solutions for FPGA designs used in applications that must satisfy safety standards
Elimination of Designed Systematic Errors
The ROI Of Early Bug Detection

- Early bug removal makes dramatic difference to project schedule

Exponential increase in cost of bug

Verification incremental, code driven
Verification top down, specification driven
Automated Design Evaluation

- Immediate code check without writing any assertions
- Catch broad range of issues early in process
- Automatic, interactive, push-button design bring-up
- Formal checks execution, not just syntax linting

Check Examples

<table>
<thead>
<tr>
<th>Structure (Easy Lint)</th>
<th>Safety Checks (Assertion Synthesis)</th>
<th>Activation (Coverage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mismatch/port /wire</td>
<td>Runtime Errors</td>
<td>Sim-Synth Issue</td>
</tr>
<tr>
<td>Signal trunc / no sink</td>
<td>Array / Range checks</td>
<td>SNPS full case</td>
</tr>
<tr>
<td>Sensitivity list issues</td>
<td>Function without return</td>
<td>SNPS parallel case</td>
</tr>
<tr>
<td>Unused signal / param</td>
<td>Signal domain checks</td>
<td>Write-write race detect</td>
</tr>
</tbody>
</table>

Dead code checks
Stuck signal (toggle test)
FSM trans and states
MORE…
3 Phase, Incremental Block Verification

- Bring-up, Code Check, Optimization

- Examine RTL Operation
  - Catch implementation errors
  - Ensure proper initialization
  - Verify key operational detail

- Expose Coding Issues
  - Find real coding errors, not false positives
  - Expose stuck signals, deadlocks, coverage holes, sim/synth issues, etc.

- Verify Design Optimizations
  - Tighten up code while ensuring equivalent function
  - Check code improvements without running simulation

“Bring-up” code without stimulus hassle  
Static operational code inspection  
Improve code without re-running regression
Check Example: Array Out-of-Bounds

- Static linting points out potential out-of-bounds access based on code
  - Unclear whether array is really accessed out of bounds during code operation
- Formal \texttt{array\_index} check:
  - Either proves that access is never out of bounds or
  - Shows simulation trace from reset with boundary violation

\begin{verbatim}
reg [2:0] i;
reg [5:0] array;
always @(posedge clk)
for(i=0 ; i<=5; i=i+1)
 array[i] <= 1;
\end{verbatim}

\textit{Lint}: This code “might” result in out of bounds access

\textit{Inspect}: Design operation does not result in an out-of-bounds access
Formal Complements the Simulation Process

EXHAUSTIVE DEEP FORMAL
Right signal output under ALL input and state circumstances

COMPLEX SCENARIO or BUG HUNTING
Test for error type X, while sequence state is A3, and frame partially aligned

CAN IT EVER HAPPEN?
Can the internal FIFO overflow in any situation?
Automated Formal Apps

- Reduce complex stimulus authoring for integration issues
- No user-written assertions – automated process
- IP integration methodology 10X faster than simulation-only
App Example: Connectivity Checking

- Highly automated, easy to set up
- Assertion-based and structural connectivity for maximum confidence
- Supports delays and conditional connectivity
- Proven on very large designs and an essential part of SoC integration
- Efficient debug of connectivity issues

Check key connections through complex structures
Assertion-based Verification
Exhaustive Formal Analysis

• Assertions specify design intent
• For each assertion:
  • Static formal analysis either proves that it can never be violated or
  • Shows a simulation trace from reset with the violation
• Any formal trace can be run in simulation for familiar debug process
Assertion-Based Verification

- Industry-standard SystemVerilog Assertions (SVA)
- High-performance/capacity with advanced engines
- Formal results back-annotated into verification plan
- Formal coverage integrated with simulation results
Elimination of Introduced Systematic Errors
FPGA Synthesis Optimization

- Key factor in design performance
- Fixed interconnect grid, LUTs, shift-registers, block RAMs, configurable DSP blocks, etc.
- Many timing, fan-out, and capacity restrictions
- Synthesis maximizes utilization by register duplication, retiming, and other sequential optimizations

FPGA synthesis tools balance logic between LUTs to improve QoR.
Synthesis and Verification Challenges

• Both synthesis and manual optimizations are error prone

Critical issues:
Incorrect wiring, user-directed logic optimizations (pragmas et. al.),
Logic retiming, pipelining, arithmetic optimizations, state initialization, …
Example Real FPGA Design Flow Bugs

Example Design Flow Issues Encountered

- Bus connection ordering
- Coincident read discrepancies
- Wrong FSM re-encoding
- Undriven or unconnected wires
- Incorrectly coded pipeline
- Incorrect BRAM parameter settings
- Clock gating for low power issues
- Place & route connection issues
- Additional, unspecified logic added
FPGA Implementation Verification

- Increasing FPGA QoR with sequential equivalence checking
- Eliminate design flow bugs and accelerate FPGA bring-up process
- Use aggressive optimizations with confidence
- Reduce post-product bug risk
Reduction of Random Error Effects
Safety-Critical Verification

• Meeting tough functional safety standards such as ISO 26262
• Efficient verification of functional safety mechanism using fault injection
• Precise diagnostic coverage with formal fault analysis accelerating existing methodologies
• Rigorous systematic verification flow

Minimize Systematic Errors
Safeguard Against Random Errors

Rigorous Verification
Verification of Safety Mechanisms

Quantification of Verification
Diagnostic Coverage

Required by Safety Standards
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
Safeguarding Against Random Errors

Random Errors: Operational issues caused by permanent or transient faults
- Examples: Latch up, bridging, single event upset/transient

Hardware-Based Safety Mechanism

Goal: Achieve desired diagnostic (fault) coverage and, therefore, protection against random failures
High-Level Design Flow with Safety in Mind

Minimize systematic errors

Safety Requirements/Goals

Safeguard against random errors

Functional Requirements

HW Specification

Design Implementation

Verification of the Safety Mechanism

Functional Verification

Safety Mechanism Implementation

Quantification of Verification

Diagnostic Coverage Verification

Tracing

Safety Analysis

Association

Area, Fault-list calculations

Covered by this Presentation

Functional Verification

HW Specification

Design Implementation

Verification of the Safety Mechanism

Quantification of Verification

Tracing

Safety Analysis

Association

Area, Fault-list calculations

Covered by this Presentation
Efficient Formal Verification with Fault Injection

**Formal Verification of Hardware Safety Mechanism**
- Write assertions to express expected behavior
- Invent a method for efficient injection of faults
- Map assertions to relevant fault scenarios

**Fault Injection App to Streamline Process**
- Simple interface to control injection of faults
- Efficient handling of huge number of fault scenarios
- Manage mapping of fault scenarios to assertions
- Seamlessly integrate safety mechanism and normal functional verification
Verification of Hardware Safety Mechanisms

Hardware Safety Mechanism

• Safeguard against random hardware faults
• Risk: Systematic hardware faults (RTL bugs) in hardware safety mechanism

Functional Verification

• Minimize systematic faults that impact random fault protection
• Avoid systematic faults – hardware behaves as intended in all relevant fault scenarios
• Challenges:
  • Mechanism inactive unless faults occur
  • ISO 26262 recommends fault injection
  • Huge number of fault scenarios – simulation cannot be exhaustive
Diagnostic Coverage of Safety Mechanisms

Challenges

ISO 26262 requires a quantitative analysis of random hardware errors and their outcome. The quantitative analysis provides key metrics to determine device safety integrity level (SIL).

New application domains, such as autonomous driving, drive higher safety integrity levels. Due to the high fault number and diversity / complexity of safety mechanisms, quantitative analysis very challenging and time consuming. Expert judgment limit reached, automation required.
Formal Fault Propagation Analysis

- App - no formal knowledge required
- Automatically classifies faults into:
  - Non-propagatable faults (safe)
  - Propagatable faults
    - Dangerous
    - Potentially detected
- Requires limited additional input
- Fault lists can be provided by the user or generated by the tool
  - Stuck-at fault model supported

\[
\text{Faults} = \text{Safe Faults}!
\]
Real-World Examples
Westinghouse Case Study

• Westinghouse using OneSpin 360 EC-FPGA on instrumentation for nuclear power stations
• Exacting verification requirements to meet SIL safety standards
• Close cooperation with Microsemi to verify complex FPGA optimizations

“The MicroSemi ProASIC3 FPGA is a core component of the Advanced Logic System (ALS), and use of the OneSpin 360 Equivalence Checker is an integral part of our FPGA development process for nuclear safety systems.”

Erik Matusek, Safety System Platform Manager, Westinghouse Electric Company, LLC
Hitachi Case Study

- Hitachi using OneSpin 360 EC-FPGA and EC-RTL on a functional safety controller for industrial facilities
- Measures for fault avoidance certified by TÜV Rheinland Industrie Service GmbH as SIL 4, the highest Safety Integrity Level of the IEC 61508 functional safety standard

“We achieved IEC 61508 SIL 4 for the fault avoidance measures during development of the functional safety controller vCOSS S-zero®, a challenging endeavor for this type of equipment. We used a number of technologies to meet SIL 4 requirements, but equivalence verification using OneSpin’s EC-FPGA and EC-RTL was indispensable”

Masahiro Shiraishi, Chief Engineer at Hitachi
Kalray Case Study

• Kalray using OneSpin’s functional safety solution on MPPA® low-latency, low-consumption, massively parallel processor arrays

• ISO 26262 flow enables usage in autonomous vehicles in addition to data center cloud infrastructure and big data analytics

“Computing hardware fault metrics and achieving targets set by ISO 26262 is challenging, but crucial to enable the application of our massively parallel many-core technology in autonomous vehicles. OneSpin is a trusted provider of apps, methodology and expertise to automate many steps of this process. Working cooperatively with its engineers smoothed our path to ISO 26262, savings months of project time.”

Camille Jalier, Kalray’s Director of Hardware R&D
Thanks for Listening!