Quantification of Formal Properties for Productive Automotive Microcontroller Verification

Holger Busch
Infineon Technologies
Overview

■ Goals
■ Application
■ Quantification Approaches
■ Onespin‘s Coverage Feature
■ Certitude
  – General set-up
  – Coupling with Onespin
■ Results
■ Conclusions
Goals

- Quality control?
  - Each single property 100% checked for all inputs!
  - **But**: specific function potentially uncovered
  - Assessment of formal property sets needed!

- Formal verification management
  - Progress control
  - Sign-Off Criteria

- Handling of mixed verification tool landscape
  - Directed & constraint driven simulation
  - Formal property checking

- Safety-compliance to ISO26262
  - Traceability of requirements
  - Reproducibility of design and verification process
Overview

- Goals
- Application
- Quantification Approaches
- Onespin’s Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin

- Results
- Conclusions
Application: AURIX® μC Family

- Multicore architecture
  - Up to three 32-bit TriCore™ CPUs (up to 300 MHz)
- Single scalable platform for target applications:
  - Powertrain:
    - Engine management
    - Transmission control
    - Hybrid and electrical veh.
  - Safety:
    - Airbag, steering, braking ASIL D (ISO 26262)
  - Driver Assistance:
    - Laser, radar, camera
  - Body:
    - Hardware security
Overview

- Goals
- Application
- Quantification Approaches
- Onespin’s Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin
- Results
- Conclusions
Quantification Approaches

- **Manual**
  - Review of formal properties

- **Formal completeness checks**
  - Onespin’s gap-free verification methodology
  - Not related to simulation coverage metrics

- **Formal witness generation**
  - Code coverage for trace: line, branch
  - Quality of witness?

- **Design mutation**
  - Onespin’s built-in coverage feature Quantify
  - Link to test-bench qualification tool Certitude
Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin
- Results
- Conclusions
Onespin 360°™ MV

- Bounded model-checker
  - Various proof engines

- Property languages:
  - ITL (Interval Language), SVA, PSL

- Consistency checker
  - Dead-code detection, ...

- Property debugger

- Coverage
  - Formal completeness checker
  - Line & branch coverage
Onespin‘s Quantify Feature

- Pre-analyses
  - Dead, constrained, redundant code identification
  - Code reachability by witness traces

- Observation coverage:
  - Formal proofs of properties with mutated code locations
  - Code location covered when proof fails

- User-guidance
  - Push-button, focussing possible

- Result
  - XML ~> UCDB-compatible
  - HTML
Onespin’s Quantify Feature

Quantification of Formal Properties for Productive Automotive Microcontroller Verification

Holger Busch

Page 11
Onespin's Quantify Feature

Quantify MDV File Result - Konqueror (on vihrc595)

```plaintext
begin
if cmd passed = '1' and smu cmd cmd j = CMD_SMU_ALARM c then
    case smu cmd arg 1 is
        when '0000' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '0001' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '0500' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '0511' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '0100' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '0101' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '0110' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '0111' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1000' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1001' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1010' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1011' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1100' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1101' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1110' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when '1111' => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
        when others => smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
    end case;
else
    smu_sw_alarm <= '0000' or '0000' or '0000' or '0000';
end if;
end process smu_sw_alarm_p;
-- Outputs
```

Quantification of Formal Properties for Productive Automotive Microcontroller Verification

Holger Busch
Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s Coverage Feature

Certitude
  - General set-up
  - Coupling with Onespin

Results

Conclusions
Certitude

- Fault-instrumentation of RTL
- Check fault detection by test-cases
Certitude Qualification Flow

- Selection: \{fault, test-case/prop\}
- Result: fail/pass, check-time

RTL*

Certitude

Simulator/ Formal Property Checker

Qualification Results

Test-cases/ Props

Scripts:
- Elaborate RTL*
- Generate fault assumption
- Check
- Write result
Certitude

- Modeling phase: RTL-code instrumentation by Certitude
  - Different fault models injected into RTL code
  - Top-level entity with additional input vector for individual activation

- Activation phase: Each test-case run once:
  - Activation: test-case stimulus activates fault condition
  - Propagation: fault visible at observation points (DUT interface)

- Detection phase: Analyses for pairs of \{fault test-case\}:
  - Detection: fail of test-case instead of pass
  - Fault-sets: $F_{\text{injected}} \supseteq F_{\text{activated}} \supseteq F_{\text{propagated}} \supseteq F_{\text{detected}}$
  - Iterative detection controlled by Certitude

- Statistical Approach by Certitude:
  - Metrics computation for statistical samples

- Application to Formal Properties
  - Iterative invocation of property checker for formal property instead of simulator for test-case
Overview

- Goals
- Application
- Quantification Approaches
- Onespin’s Coverage Feature

Certitude
- General set-up
- Coupling with Onespin

Results

Conclusions
Certitude <-> Onespin

Iterative procedure:
- Let Certitude select:
  - Property $P$ from set of qualification properties
  - Fault $c$ from current set of non-detected faults
- Add fault assumption to regular property

Regular Property $P$:
\[ \text{ass}(P) \vdash \text{com}(P) \]

Property $P$ with enabling of fault $c$:
\[ \text{1hot}(f) \land f(c) = 1, \quad \text{ass}(P) \vdash \text{com}(P) \]

- Check fault-$c$-enabled Property $P$ in property checker
- Return proof result + run-time to Certitude
  - Fail: fault $c$ detected by Property $P$
- Repeat until Certitude is finished:
  - All faults detected
  or
  - All \{fault,property\}-pairs exercised
Certitude <-> Onespin

**Challenges:**

- Large number of \{fault, property\}-pairs to be formally checked
  - Estimated full qualification time: \( t_{\text{qual}} = 0.5 \times n_{\text{faults}} \times n_{\text{props}} \times t_{\text{check}} \)
    
    Example:  \( n_{\text{faults}} = 5000, \ n_{\text{props}} = 200, \ t_{\text{check}} = 5 \text{ min} \rightarrow t_{\text{qual}} \approx 9.5 \text{ years} \)

- Repeated invocation of property checker causes overhead
  - Re-elaboration or loading DUV model
  - Loading properties with current fault assumption

- Instrumented RTL-design not always clean:
  - Combinational signals become latches for some fault classes
  - Oscillating signals

- Proof-time for individual check often differs from normal proof
  - More powerful provers invoked if fast prover fails
  - Different provers for counterexample generation
Certitude <-> Onespin

- **Reduction of check times**
  - Minimization of set-up time before check
    - Keep property checker session open
      - wait for new task sent by Certitude
      - just load new fault constraint
    - Simultaneous property checks (≤ available tool licenses)
  - Selection of proof engines
    - Evaluation of log files

- **Reduction of pass-checks**
  - Theoretical minimum: $n_{\text{checks}} = n_{\text{faults}} \left( \leq 0.5 \times n_{\text{faults}} \times n_{\text{props}} \right)$
  - Selection of {fault, property}-pairs essential
    - Certitude‘s heuristics: Analyze previous results
    - Human knowledge: Relate properties to code partitions
    - Analyses in Formal Property Checker
Overview

- Goals
- Application
- Quantification Approaches
- Onespin's Coverage Feature
- Certitude
  - General set-up
  - Coupling with Onespin

- Results
- Conclusions
## Results

<table>
<thead>
<tr>
<th>Module verification</th>
<th>Quantify</th>
<th>Certitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Locs (VHDL)</td>
<td>Props</td>
</tr>
<tr>
<td>1</td>
<td>25563</td>
<td>85</td>
</tr>
<tr>
<td>2</td>
<td>27374</td>
<td>157</td>
</tr>
<tr>
<td>3</td>
<td>57168</td>
<td>253</td>
</tr>
</tbody>
</table>

*) ~ 80% covered, no progress
## Comparison

<table>
<thead>
<tr>
<th>Quantify</th>
<th>Optimized Certitude Qualification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal code coverage by internal mutations</td>
<td>Explicit fault injection in RTL design</td>
</tr>
<tr>
<td>Closed, indirect controllability</td>
<td>Flexible, full controllability and extensibility</td>
</tr>
<tr>
<td>Fast for simple parts, potentially very long computations for more difficult parts; acceptable efficiency for small – medium designs; Open code regions for biggest design, but ongoing improvements by Onespin</td>
<td>Scales to big designs</td>
</tr>
<tr>
<td>Restartable, longer setup time</td>
<td>Fast restartability</td>
</tr>
<tr>
<td>Onespin session + prover licenses</td>
<td>Onespin session + prover licenses + Certitude license dynamic parallelisation</td>
</tr>
<tr>
<td>Code coverage stronger than simulation metrics: formal proofs of observability</td>
<td>Merge with simulation qualification 1:1</td>
</tr>
<tr>
<td>Product quality</td>
<td>Packaging of scripts required for wider usage</td>
</tr>
</tbody>
</table>
Overview

- Goals
- Application
- Quantification Approaches
- Onespin‘s coverage feature
- Certitude
  - General set-up
  - Coupling with Onespin

Results

Conclusions
Conclusions

- Two feasible quantification approaches for FPC!
- Both manage big modules with several 10 k loc
- Quantification results largely comparable
- Lots of FPC licenses used
- Long-running properties disadvantageous

**Onespin‘s Quantify:**
- Efficient, but closed
- Metric similar to simulation code coverage, but stronger

**Certitude-Onespin coupling:**
- Open for optimizations & configuration
- Exactly same metric for simulation
Thank you!