An Efficient Methodology for Mutation-Coverage-Collection of Formal-Property-Checking

Holger Busch

Infineon Technologies AG
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions
Mutation Coverage

• Goal: Safeguard verification quality
  • Provide completeness metrics and sign-off criteria
  • Measure verification progress
  • Check whether function of each statement is verified

• Approach: Systematic fault insertion
  • Instrument design: inject functional mutations + multiplexors
  • Iteratively activate faults and collect detections by regression
  • Detection: test case / property failure
EDA-Tools Supporting Mutation Coverage

• Quantify - Onespin
  • Integrated in formal-property-checker
  • Instrumentation of model, line-based
  • Push-button

• Certitude - Synopsys
  • Separate from verification tools
  • Usable with any simulator or formal property checker
  • Integration scripts required
  • Configurable instrumentation of HDL-design

• Today’s topic: Integration of Onespin’s FPC with Certitude
For each detection-run, Certitude selects a pair of fault–testcase: high number of combinations!
Configuration of Design Instrumentation

• Code regions to be instrumented

• Fault Categories:
  • Replacement of right-hand side of assignments
    • Free-variable inputs, negation, operator replacement, operand swaps
  • Replacement of Block Conditions
    • Tied to true or false, negation
  • Signal distortion
    • Tied to 0 or 1, negation
Instrumentation Example

```
321  cpu_idle_ack_s <= '1' when (state /= run and
322    pmcsrx_reqslp_s = "11" and
323    pmswctrl_iradis_i = '1') else
324  cpu_idle_ack_i;
```

<table>
<thead>
<tr>
<th>Fault ID</th>
<th>Fault Type</th>
<th>Fault In Report</th>
<th>Status</th>
<th>Detected By Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>764</td>
<td>ConditionFalse</td>
<td>✔</td>
<td>Detected</td>
<td>sm_fsm</td>
</tr>
<tr>
<td>765</td>
<td>ConditionTrue</td>
<td>✔</td>
<td>Detected</td>
<td>sm_fsm</td>
</tr>
<tr>
<td>766</td>
<td>NegatedCondition</td>
<td>✔</td>
<td>Detected</td>
<td>sm_fsm</td>
</tr>
</tbody>
</table>

With the fault 764 of type 'ConditionFalse', the code:

```
321  cpu_idle_ack_s <= '1' when (state /= run and
322    pmcsrx_reqslp_s = "11" and
323    pmswctrl_iradis_i = '1') else
```

is changed into:

```
321  cpu_idle_ack_s <= '1' when false else
```
cpu_idle_ack_s <= '1' when ((cerfaultenable518to777(764) = '1') and false)
   else '1' when ((cerfaultenable518to777(766) = '1') and not boolean (((state /= run) and (pmcsrx_reqslp_s = "11")) and (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(768) = '1') and (((state /= run) and (pmcsrx_reqslp_s = "11")) or (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(770) = '1') and (((state /= run) and (pmcsrx_reqslp_s = "11")) and (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(771) = '1') and (((state = run) and (pmcsrx_reqslp_s = "11")) and (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(772) = '1') and (((state /= run) and (pmcsrx_reqslp_s /= "11")) and (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(773) = '1') and (((state /= run) and (pmcsrx_reqslp_s = "00")) and (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(774) = '1') and (((state /= run) and (pmcsrx_reqslp_s = "01")) and (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(775) = '1') and (((state /= run) and (pmcsrx_reqslp_s = "10")) and (pmswcr1_iradis_i = '1')))
   else '1' when ((cerfaultenable518to777(776) = '1') and (((state /= run) and (pmcsrx_reqslp_s = "11")) and (pmswcr1_iradis_i /= '1')))
   else cer_tbq_FreeSignalCopy_767_0_cpu_idle_ack_s when ((cerfaultenable518to777(767) = '1') and (((state /= run) and (pmcsrx_reqslp_s = "11")) and (pmswcr1_iradis_i = '1')))
   else '1' when (((cerfaultenable518to777(765) = '1') and false) or (cerfaultenable518to777(770) = '1') or (cerfaultenable518to777(771) = '1') or (cerfaultenable518to777(772) = '1') or (cerfaultenable518to777(773) = '1') or (cerfaultenable518to777(774) = '1') or (cerfaultenable518to777(775) = '1') or (cerfaultenable518to777(776) = '1') or (cerfaultenable518to777(767) = '1') or (cerfaultenable518to777(768) = '1') or (cerfaultenable518to777(769) = '1') or (cerfaultenable518to777(770) = '1') or (cerfaultenable518to777(771) = '1') or (cerfaultenable518to777(772) = '1') or (cerfaultenable518to777(773) = '1') or (cerfaultenable518to777(774) = '1') or (cerfaultenable518to777(775) = '1') or (cerfaultenable518to777(776) = '1'))
   else cer_tbq_FreeSignalCopy_777_0_cpu_idle_ack_s when ((cerfaultenable518to777(777) = '1') and true)
   else cpu_idle_ack_i;
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions
Formal Certitude Flow
Preparation Steps

• User specifies code regions to be instrumented and properties
  • Exclusion of pre-verified libraries, generated code, re-used components

• Automatic steps:
  • Certitude configuration and invocation
  • Instrumented RTL design loaded into Onespin
  • Instrumented properties loaded
  • Sanity-proofs with 0-fault assumption:
    • Failing properties excluded
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions
User Control

- Started by user with optional parameters for detection control
  - Property subset to be used for qualification
  - Target code regions with instrumented but not yet detected faults
- Generated default configuration file intermediately adjustable by user
  - Limits for time, memory, parallelism
  - Maximum number of iterations (default: unlimited)
  - Verbosity
Automatic Iterative Procedure

• Execution of consecutive rounds:
  • Selection of current property sub-set: ranking by run-times
  • Adjustment of fault-enabling assumptions
  • Qualification proofs
  • Result evaluation

• Termination
  • No undetected faults left
  • All qualification properties proven or excluded by configured time-limit
  • User-specified number of iterations reached
Fault-Enabling Assumption

- Specifies fault-set addressed in next qualification proofs
  - Subset of original target faults not yet intermediately detected since start

Regular Property $P_j$:

$$\text{ass}(P_j) \vdash \text{com}(P_j)$$

Property $P_j$ with enabling of fault $f_i$:

$$\text{lhot}(fv) \land fv(i) = 1,$$

$$\text{ass}(P_j) \leftarrow \text{com}(P_j)$$

Fault activation vector: only one fault enabled in each check

Formal flow: Selection of fault yielding failure by formal prover
Detection Proofs

• Automatic submission of proof jobs out of Onespin

• Evaluation of results:
  • Proven properties:
    • None of currently addressed faults detectable
    • Remove from qualification property set
  • Disproven properties:
    • Collect detected faults and subtract from fault set
    • Record proof times
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions
Objectives

• Maximization of detection speed
  • Reduction of model / proof complexity
  • Let fast-running properties detect faults first
  • Avoid useless attempts
  • Increase parallelism
  • Focus on new detection goals, re-use previous results

• Minimization of overall resource consumption
  • LSF-hosts heavily used by competing jobs
Prover Selection

• Prover groups in Onespin:
  1. Search from arbitrary states
     • Counterexample may be unreachable
     • Hold-result valid in complete state space including unreachable part
  2. Search from reset state
     • Expensive or unfeasible if huge number of cycles required before assumption state

• Detection yielded from fail-result
  • Reachable failure impossible for some properties proven by 1.

• Detection proofs of 1-properties are run with 1-provers
Focusing

• Specific code regions, function, and property subsets are related
• User can specify relations
• Local detection accelerates qualification:
  • Additional or enhanced property targeted at specific uncovered code
  • Minimum wait-time until feed-back whether enhancement sufficient
Model Trimming

• Problem:
  • Instrumented model much more complex
  • Complex properties potentially unusable

• Approach:
  • Instrumented design: additional input vector for fault-enabling
  • Re-compilation with Onespin-option for tying fault-enabling input-bits

• Procedure:
  • Re-compilation for current fault-subset
  • Percentage of detected faults automatically triggers model trimming

• Effect:
  • Model continuously reduced with detection progress
  • Advantageous for postponed long-runners!
Super-Parallelization

• Several independent qualification sessions with disjunct fault subsets
Merging Results

• Separate qualification sessions
  1. Same Certitude instrumentation:
     • Onespin-qualification results directly merged and imported into Certitude
  2. Same design version, but different Certitude instrumentations
     • Merged Certitude instrumentation
     • Fault-mapping based on fault attributes
     • Merging detections of mapped faults in Onespin and Certitude
Inheritance

• Change requests until tape-out
  • Few design code affected
  • New instrumentation
  • New or modified formal properties

• Restart of qualification from scratch avoided
  • Fault-mapping
  • Tentative re-use of previous detections in directed-qualification procedure
  • Only remaining undetected faults addressed by regular detection procedure
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions
No Prerequisites

• Flow started in normal Onespin session with proven properties
• Few simple commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
<th>Shell</th>
</tr>
</thead>
<tbody>
<tr>
<td>cqm $props $incl $excl $qfn</td>
<td>Prepare instrumentation</td>
<td>Onespin-TCL-shell</td>
</tr>
<tr>
<td>cqd $faults $props</td>
<td>Run detection rounds</td>
<td>Onespin-TCL-shell</td>
</tr>
<tr>
<td>cqd $cert_db</td>
<td>Run detection directed by previous Certitude database</td>
<td>Onespin-TCL-shell</td>
</tr>
<tr>
<td>cqdp $n</td>
<td>Start parallel qualification sessions</td>
<td>Onespin-TCL-shell</td>
</tr>
<tr>
<td>cqa $new_props</td>
<td>Augment qualification property set</td>
<td>Onespin-TCL-shell</td>
</tr>
<tr>
<td>cqdm $qdirs</td>
<td>Merge parallel subsessions from qualification directories</td>
<td>Onespin-TCL-shell</td>
</tr>
<tr>
<td>mcq $cert_dbs</td>
<td>Merge results from several Certitude databases</td>
<td>Any TCL-shell</td>
</tr>
<tr>
<td>codvis</td>
<td>Visualize Onespin detection status in Certitude HTML report</td>
<td>Linux-command-shell</td>
</tr>
</tbody>
</table>
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions
Summary

- Valuable structural completeness metrics for formal
- Fast detection progress
- Minimized complexity
- Automation: ease of use
- Status import into Certitude at any time
- Continuous improvements by wide experience
- Mutation coverage necessary, but not sufficient
  - Deviations from specification not captured
Agenda

1. Introduction
2. Preparation Phase
3. Detection Phase
4. Performance Improvements
5. Usage
6. Summary
7. Questions