Relative Functional Qualification of a Reusable Testbench

Satwinder Singh
Infineon Technologies UK Ltd

April 28, 2014
DVClub UK
Agenda

- Introduction to Functional Qualification (FQ)
- Traditional Approach to FQ
- Compatible Verification Environment
- The New Flow – Relative FQ
- Results
- Summary
The Basics

Mutation Testing & Certitude
Functional Qualification of a VE

Activation: Exercising part of design that contains bug
Propagation: Moving an activated error to observable point
Detection: Making a comparison between design and reference behaviour and then reporting the bug.
Verification Improvement Methodology using Certitude

- Typically run towards the end of the verification project
  - Design is complete & stable
  - Testbench is mature, stable and all testcases complete
  - Fair degree of coverage targets reached

<table>
<thead>
<tr>
<th>Main Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Run model and activation phase</td>
</tr>
<tr>
<td>2) Analyze activation results</td>
</tr>
<tr>
<td>3) Run detection phase overnight</td>
</tr>
<tr>
<td>4) Analyze a Non-Detected fault</td>
</tr>
<tr>
<td>4) Improve VE</td>
</tr>
<tr>
<td>5) Run activation and detection phases overnight</td>
</tr>
<tr>
<td>4) Analyze a Non-Detected fault</td>
</tr>
<tr>
<td>4) Improve VE</td>
</tr>
<tr>
<td>6) High-quality Verification</td>
</tr>
</tbody>
</table>

Until all Non-Detected faults for this run are processed

Until VE matches quality requirements or there are no more Non-Detected
Challenges of Traditional Approach

• RunTime – Fairly high
  – Complexity of the design
  – Number of testcases
  – Propagation/Detection holes encountered
  – About 3 weeks on our design
• Debug Effort – Very High
• Project Deployment Stage – Quite Late when Design & TB are stable
• Frequency of runs – Once of twice in the whole project
The Background

A Compatible Verification Environment
Compatibility of the Verification Environment

- Often a requirement to resurrect an old VE
  - For customer requirements to explore a specific behaviour
  - Validate if a bug found on current projects exists on a previous release
  - Redeliver an old project
- Historically achieved by taking snapshots of the VE in the past but has issues –
  - Changes to tooling or IT Infrastructure
  - Difficulty in reproducing the required stimuli
  - Loss of familiarity of the old VE
- Solution – Have a VE that is backwards compatible so the current VE can be used to sign-off earlier releases.
Reuse and Compatibility

- Testbenches not written from scratch on every project but are matured and reused.

Previous Project

- Base Testbench
- Reuse
  - Delta FX added
  - Configurability added
  - Improvements (Code/Scripting)
  - Improvements (Data/Job Management)
  - Tool changes
  - Methodology/Language Changes

Compatibility

Next Project

- Ported Testbench

- Backwards Compatibility => Current/Ported VE is as good as earlier ones at finding bugs => “Forwards Compatibility”
- Provided there exists a mechanism/metric to compare the quality of the two VEs
The New Flow

Checking Testbench Equivalence
Relative Functional Qualification

• Comparing Quality or the Bug finding capability of two TBs
• Mitigates the necessity of carrying out a full Functional Qualification
  – Provided the base TB used in the comparison is a “Healthy” TB

Healthy Testbench –
  – A full functional qualification has been carried out on the base TB on the previous project with satisfactory results
  Or
  – The base TB has been used to successfully tapeout projects in the past with proven silicon functionality thus creating confidence
Novel Approach for TB Quality Comparison

- Compatibility & “Forwards Compatibility” – basis for Quality Comparison

- Introducing Certitude Metric Mode
  - Mode for statistical estimation of Propagation & Detection scores
  - Application of statistical algorithms to deduce a subset of faults – Metric Sample
  - Accuracy of results driven by “Margin of Error” (MOE)
  - Upto 1% MOE (99% accuracy) configurable, 2-5% generally suitable
  - Metric Sample – A good representation of overall fault set
Novel Approach for TB Quality Comparison

New Metric for TB Quality Comparison!!
## Results

- New Flow deployed on a signification change of our Verification Environment

<table>
<thead>
<tr>
<th>Result</th>
<th>Golden Database (3% MOE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Creation time (Instrumentation + Activation + Metric run)</td>
<td>&lt; 2 days (One time effort)</td>
</tr>
<tr>
<td>Metric results - Global Score:</td>
<td>70.33%</td>
</tr>
<tr>
<td>Faults in metric sample:</td>
<td>900</td>
</tr>
<tr>
<td>Detected Faults:</td>
<td>626</td>
</tr>
<tr>
<td>Relative Functional Qualification run time</td>
<td>1 hr (Nightly/Weekly run)</td>
</tr>
<tr>
<td>Number of Detection Holes (Bugs)</td>
<td>3 + 3</td>
</tr>
</tbody>
</table>
Summary
## Comparison with Traditional Flow

<table>
<thead>
<tr>
<th>Measure</th>
<th>Traditional Flow</th>
<th>New Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>3+ weeks on our design</td>
<td>1 - 1.5 hrs</td>
</tr>
<tr>
<td>Debug Effort</td>
<td>Very high. It can be very laborious to root cause propagation or detection holes in a complex design or testbench.</td>
<td>High, however much simpler compared to traditional flow because of the presence of a successful reference detection.</td>
</tr>
<tr>
<td>Project Deployment Stage</td>
<td>Late in the project cycle when design &amp; testbench are mature</td>
<td>Beginning of the project</td>
</tr>
<tr>
<td>Frequency of runs</td>
<td>Once or twice in the whole project</td>
<td>Nightly / Weekly</td>
</tr>
</tbody>
</table>

- Speedup in excess of **200 times** achieved!!
Proposed Use Model

Previous Project
- Base Testbench Functionality

Next Project
- Base Testbench Functionality with added Configurability + Any Improvements
- Delta Testbench Functionality

Testbench Reuse

- Qualified by proposed (new) flow
- Start Qualification from the beginning of the project
- Run Nightly

- Optionally qualify the whole testbench using traditional flow towards the end of the project
- Qualified by Traditional Flow
- Should be reused on next project (qualified by the new flow)
Conclusion

• New Flow ensures TB quality consistently maintained throughout the project, not just at the end
• 200 times faster than traditional approach (just a bit less accurate)
• New Flow relies on Compatibility of the VE
• Found 6 bugs on our VE improvements (early in the project cycle)
Thank You
Backup Slides
Introduction to Mutation Testing

- Old software approach to test the quality of testing programs
- More recently applied to Hardware Verification
- Mutation – Simple fault injected into a design
  - Simple fault injected into a design
    - Changing an operator in a HDL Expression
  - Not necessarily represent real design bugs
- Mutation Testing or “Functional Qualification” for a VE relies on the “Coupling Effect” hypothesis
  - Test that detects simple faults in a design is sensitive enough to distinguish typical complex design bugs
Functional Qualification with Certitude – An Overview

- Models a selection of simple faults and measures the ability of the VE to find complex design bugs
- Exact nature of mutations depend on the chosen fault model.
- Certitude FQ Phases -
A new configuration / project could be verified just by adding a new configuration file !!
Algorithm

- Creating Golden DB using Base VE

1. **Prepare Design**
   - Instrument the design
   - Run Activation
   - activate

2. **Estimate Detection Profile**
   - Run Metric Mode
     - `computemetric -marginoferror = 0.03`

3. **Full Detection of Metric Sample**
   - Save Metric Sample
     - `faultlist -metricsample`
     - `userdataadd -userdatakey=METRICSAMPLE -userdatavalue=\{faultlist -metricsample\}`

4. **Run Detection on Metric Sample**
   - `detect -updateresults`

5. **Save Detected Fault List**
   - `userdataadd -userdatakey=DET_FLT_LIST -userdatavalue=\{faultlist -status=Detected\}`

6. **Save the Database**
   - `dbdump -directory=cer_compatibility_golden.db`
Algorithm

• Qualification Run (Modified VE)

- **Reload Golden DB**

- **Detection Profile of Golden Run**

- **Testcase compatibility on modified VE**

  - **Extract Detected Faultlist**
    - `set DET_FLT_LIST [userdatainfo -userdatakey=DET_FLT_LIST]

  - **Create Fault-Testcase Pairs**
    - `foreach flt $DET_FLT_LIST {
        set fltTestMap($flt) [faultinfo -fault=$flt -detectedby]
        set testcase [faultinfo -fault=$flt -detectedby]
        lappend fltTestList $testcase
        set tmpList [list]
        lappend tmpList $flt $testcase
        lappend fltTestPairList [list $flt $testcase]}

- **Collect Detection Profile on new VE**
  - `checkdetection - faulttestcasepairlist=$fltTestPairList`

- **Run all testcases on non-mutated design with new VE**
  - `testscript -noinstrumentation -testcaselist=$fltTestList`

- **Restore the Database**
  - `dbload -directory=cer_compatibility_golden.db`
Algorithm (Post Qualification Checks)

1. All testcases pass non-mutated design & new VE?
   - Yes
   - Testcase Not Compatible with new VE
   - Fix Non-Compatibility / Discard Test

2. 100% Detection Fault ↔ Testcase Pairs?
   - Yes
   - Potential Bug in New VE
   - Debug
   - No

3. Quality Checks Passed
Fault ↔ Testcase Pair Binding

• Critical for the flow to work
• Needs to be sufficiently strong
• Same Testcase should still detect its corresponding fault on modified VE
• Achievable when Testcase behaviour hasn’t changed in the new environment.
  – Testcases reproducible by a seed when generated randomly
  – Testcases generated offline independent of the VE
Limitations & Future Possibilities

• Flow relies on the existence of a High Quality Base Testbench

• High Strength of Fault $\Leftrightarrow$ Testcase binding needed
  – Existence of offline random generator helps make it strong
  – May be weaker in case of interactive Test Generation
  – Could be enriched by associating multiple Tests to a Fault

• Statistical Estimation
  – Results only accurate within the Margin of Error
  – Minimum possible MOE is 1%, but runtime increases exponentially