Coverage-Oriented Verification of Banias

Alon Gluska

Intel Corporation

Haifa, Israel

Agenda

- Introduction
- Coverage-Driven vs. Coverage-Oriented verification
- Functional coverage in Banias and results
- Coverage studies and conclusions
- Summary

Introduction

- Functional coverage was widely applied in Banias verification
- We used coverage-oriented approach
  - A practical alternative to coverage-directed
- Revealed fewer RTL bugs than expected
  - Yet, silicon was exceptionally clean
- Evaluation revealed significant impact beyond original intentions
  - And conclusions for the future

Agenda

- Introduction
- Coverage-Driven vs. Coverage-Oriented verification
- Functional coverage in Banias and results
- Coverage studies and conclusions
- Summary

Coverage-Driven Verification

- Coverage enables better utilization of resources
  - Functional coverage is derived from spec
  - Related to quality, but difficult to define and measure
- In Coverage-Driven Verification (CDV), coverage drives verification start-to-end
  - Coverage tasks are defined a priori
  - Verification is steered towards coverage holes
  - Motivation: most cases are hit by random testing
- However, CDV is frequently impractical
  - Limited design knowledge in early stages
  - Design instability impacts coverage results
  - In early stages, focus is given to bug finding

Coverage-Oriented Verification

- A practical trade-off to Coverage-Driven
- Start with verification aimed to hit broad functionality
  - Quickly detect easy-to-find bugs
  - Use ‘light’ test plans
  - Execute random and semi-random tests
- When bug rate drops:
  - Coverage drives the completion of verification
    - Feedback for testbenches, test plans, tests
    - Steer verification resources accordingly
  - Detailed, coverage-oriented test plans
    - Explicitly specify coverage spaces and targets

Agenda

- Introduction
- Coverage-Driven vs. Coverage-Oriented verification
- Functional coverage in Banias and results
- Coverage studies and conclusions
- Summary

Verification of Banias CPU

- First microprocessor designed solely for mobile computing
  - Targets performance, form factors, battery life
  - 80M transistors, 350 functional blocks
- Modular verification
  - Design was partitioned into 6 clusters
  - Verification was done mostly in Cluster Test Environments (CTEs)
  - Tests were mostly random / directed-random tests
  - 62% of RTL bugs found at this level
- Full-Chip for architectural conformance, cross-cluster features and uArch stressing
- Limited formal verification

Functional Coverage in Banias

- Coverage effort reflected verification methodology
  - Mostly performed at cluster level
  - Full chip for cross-cluster and additional confidence
- Effort began in middle of verification period
  - Most random templates were already implemented
- Internal tools for instrumentation, measurement, and analysis
- Number crunching:
  - 1.3M micro-architectural conditions
    - Reached >95% with respect to targets
    - Targets consist on both density and distribution
  - 15% new tests for coverage holes
  - ~12% of verification resources

Coverage Results

- 19 RTL bugs, out of 365 (5.2%) in last 6 months
  - Not uniformly distributed across all clusters
- In most clusters, coverage was applied:
  - After more-than-moderate RTL cleanup
  - During intensive IA32 compliance testing
- Coverage was not applied in areas of lower criticality
  - Testability, performance monitors, power
- Cluster test plans and CTEs were found to be incomplete
  - Holes covered by IA32 compliance testing
- Coverage monitors were too detailed in many places
  - Hampered focus on riskier areas

Agenda

- Introduction
- Coverage-Driven vs. Coverage-Oriented verification
- Functional coverage in Banias and results
- Coverage studies and conclusions
- Summary

Conclusion 1: Impact of coverage

The number of bugs exposed in RTL is prime factor in verification. However:

- **Impact of coverage is beyond raw number of bugs**
  - Provides feedback for accuracy and effectiveness of random testing
  - Quantitative indicator of design quality
- **In Banias:**
  - Half of coverage bugs were “hard-to-find”
  - Coverage enforced study of low-level uArch details
  - Coverage analysis enabled trimming expensive simulation cycles

Conclusion 2: Focus coverage

- Coverage is not uniformly effective over verification tasks
- Focus and prioritize coverage towards the more complex and risky features
- Additional considerations to reduce/drop coverage:
  - High controllability
  - Effective random testing
- In Banias:
  - Coverage was not as focused as needed
  - We should have:
    - Decided up-front where to skip coverage
    - Dropped coverage when becomes ineffective

Conclusion 3: Coverage volume

- An N-dimensional coverage space can be easily produced by all permutations of N vectors
  - Resulting in huge spaces
- Such spaces are frequently very hard to cover
  - With no indication for issues
- Don’t specify what you cannot cover
- In Banias:
  - We defined manageable subsets of original domains after wasting time with ineffective analysis

Conclusion 4: Timing coverage tasks

- Coverage should be aimed towards the hard-to-reach cases
  - In early stages, bugs are easy to find by random tests
- Start coverage just before failure rate drops, when:
  - Design becomes stable
  - Cost of bugs crosses a threshold
  - Crafting tests for corner cases is required
  - Verification engineers have acquired deep knowledge
- In Banias:
  - We started coverage slightly late in almost all clusters

Conclusion 5: Friendly test plans

- Need for coverage-aware test plans
  - Formally specify coverage spaces
  - Refer to well defined uArch events
  - Define the coverage targets
  - Define the relative importance of each monitor
- Supports clear and unambiguous interpretation
  - In reviews and implementation
- In Banias:
  - Test plans that served well for tests were vague and incomplete for coverage

Conclusion 6: Feedback to test gen

- Random testing should be biased
  - To enable hitting corner cases
- Use coverage as a feedback to improve test generation quality
  - Can be used to identify inaccuracies or improper distribution
- In Banias:
  - Coverage revealed bugs and guided tuning in all cluster testbenches

Summary

- Coverage-oriented verification is a practical trade-off
  - Focus shifts gradually to coverage
- Used in verification of Banias
  - Yield fewer bugs than expected
  - Bugs were not distributed uniformly
- Conclusions to make coverage more effective
  - Impact of coverage is beyond number of bugs
  - Focus and prioritize coverage
  - Don’t specify what you cannot cover
  - Start coverage just before bug rate drops
  - Test plans should be coverage-aware
  - Use coverage to improve test generation