COMS31700 Design Verification:

Are we there yet?

(The back-end of the verification cycle)

Kerstin Eder

(Acknowledgement: Avi Ziv from the IBM Research Labs in Haifa has kindly permitted the re-use of some of his slides.)

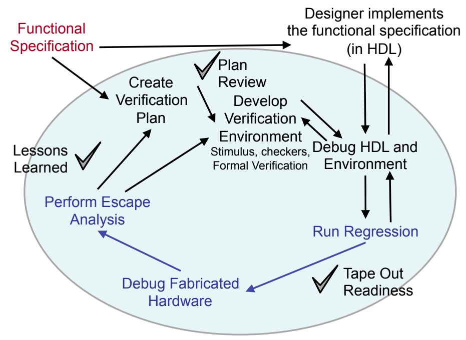
University of BRISTOL

Department of COMPUTER SCIENCE

Outline

- The verification cycle - revision
- Coverage Closure
- Analysis and adaptation
  - (Coverage analysis)
  - Failure analysis
- Regression
- Tape-out readiness
- Escape analysis

The Verification Cycle



My Environment Is Ready. Now What?

- More functionality is added to the design
  - And therefore, to the verification environment
- Mature enough design is progressed to the next level in the design hierarchy
  - Unit to core to chip to system
- Bugs are being discovered and fixed
  - And bug fixes need to be verified
- The implementation of the verification plan continues
  - Closing holes in coverage
  - Updating the verification plan itself as needed
- Regression is being executed to ensure everything still works

Coverage Closure

Coverage Closure

Coverage closure is the process of:

- Finding areas of coverage not exercised by a set of tests.
  - Coverage Holes!
- Creating additional tests to increase coverage by targeting these holes.
  - Beware: Aim to “balance” coverage!

Controllability Problems

If the cases to be hit contain internal states/signals of the DUV, directed tests that exercise all combinations are hard to find.

- Processor pipeline verification: Control logic, Internal FSMs

- Generate biased random tests automatically. [RTPG]
  - ISG **(Instruction Stream Generator)**
  - Typically tests are filtered to retain only those that add to coverage.
  - Coverage analysis indicates **hard-to-reach** cases.
  - Don't waste engineers time on what **automation** can achieve.
- Combine automatically generated stimulus with coverage.
- Gives rise to **Coverage DRIVEN Verification Methodology**

BUT:

- **Hard-to-reach cases (may) need manual attention.**
  - Bias tests towards certain conditions or corner cases.
  - Supplying bias requires significant engineering skill.
    - Often only trial-and-error approach.

7

Biasable Random Test Generation

[A Adir, E Almog, L Fournier, E Marcus, M Rimon, M Vinov and A Ziv. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. In IEEE Design & Test of Computers, 2004.]

- **Test template description language**
  - expressiveness of programming language
  - virtually unlimited control over events to be generated
- **Architectural modelling framework**
  - building blocks at high enough level to describe processors
    - VLIW architectures
- **Generation engine based on constraint satisfaction**
  - generic constraint solver customized for pseudo-random test generation

8

80/20 Split

In practice: 80/20 (20/80) split wrt coverage progress.

Good news:)

- 80% of coverage is achieved (relatively quickly/easily) driving randomly generated tests.
- This takes about 20% of total time/effort/sim runs spent on verification.

Bad news:(

- Gaining the remaining 20% coverage,
  - i.e. filling the remaining coverage holes (which often needs to be done manually and requires a lot of skill plus design understanding),
- can take as much as 80% of the total time/effort/sim runs spent on verification.

9

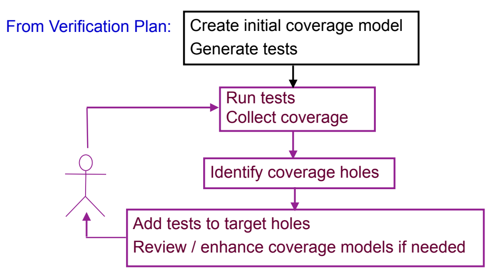
Benefits of Coverage DRIVEN Verification **Methodology**

Benefits:

- Shortens implementation time
  - (Initial setup time)
  - Random generation covers many "easy" cases
- Improves quality
  - Focus on goals in verification plan
  - Encourages exploration/refinement of coverage models
- Accelerates verification closure
  - Refine/tighten constraints to target coverage holes

10

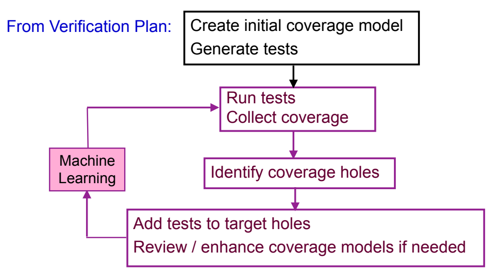
Coverage DRIVEN Verification **Methodology**



Current research: How can we automate this further?

11

Coverage DIRECTED Test Generation



Current research: How can we automate this further?

12

CDG: Coverage DIRECTED Test Generation

How can we make better use of coverage data to **automate** stimulus generation?

Latest Research:
Coverage DIRECTED (stimulus/test) generation [IBM]

- BY CONSTRUCTION
  - Require description of design as FSM.
  - Use formal methods to derive transition coverage.
  - Automatically translate paths through FSM to test vectors.
  - Fall over in practice: FSMs are prohibitively large!
- BY FEEDBACK
  - (Exploit Machine Learning techniques)
  - GAs/GP - Need to find suitable encoding (e.g. of instructions).
  - Bayesian Networks - Need to design and train BN.
  - Data Mining in coverage spaces

No significant breakthrough in CDG yet!

13

Summary: Coverage Closure

- Verification Methodology should be **coverage driven**.
- **Automation:** Research into coverage directed test generation
- **Delays in coverage closure** are the main reason why verification projects fall behind schedule!

14

Analysis and Adaptation

Regression

Analysis and Adaptation

- Building a good verification plan is the first step for successful verification
  - But, it is not enough!
- Need to constantly:
  - Monitor the verification process
  - Analyze the observations
  - Adapt to address issues identified by the analysis
- Three basic levels of adaptation
  - Change the way the verification environment is activated
  - Change the verification environment
  - Change the verification plan

16

Two Types of Analysis

1. Coverage analysis
   - Was included in the lectures on coverage.
2. Failure analysis

17

Failure Analysis

Skip to Regression

Failure Analysis

- During execution of the verification plan (many) failures are observed
- This is not a bad phenomena
  - Remember that the goal of the verification process is to identify faults in the DUV
- The goal of failure analysis is to understand failures, their causes, their relation to one another, and their relation to the verification process

Failures and Faults

- **Failure** – an observed DUV behavior that violates the specified behavior
- **Fault** – the root cause of a failure
- There can be a many-to-many relationship between faults and failures
  - Mishandling of overflow in the input FIFO can cause:
    - Lost commands in the output port
    - Bad data in the output port
  - Bad data in the output port can be caused by:
    - Mishandling of overflow in the input FIFO
    - Bad selection in the output selector

How Failures Are Detected

- Inspection and code review
- Output of formal verification tools or other static analysis tools, such as lint
- Activation of response checkers during simulation
- Analysis of coverage data
- Visual observation of application misbehavior

Types of Failure Analysis

- Detailed failure analysis
  - Understand the cause and effects of failures and faults on the design, environment, verification process and more
- Statistical failure analysis
  - Identify trends, provide prediction

Detailed Failure Analysis

- The outcome of the analysis
  - The failure is understood and recorded
  - The failure is resolved
  - The verification plan and process are adapted
  - Lessons learned for the future
- Note: In most cases failure analysis—and especially the last two items—are simple and the outcome of the analysis is that we found a failure and a fault when and where expected and because we are doing our job the right way.

Understanding the Failure

- The goal is to understand the scope and severity of the failure and how the failure can be recreated
- Provides useful information for debugging and other parts of the failure analysis
  - Simplify and generalize the failure conditions
    - Find simper settings / stimuli that recreate the failure
    - Find necessary and sufficient conditions for the failure
  - Localize the fault in terms of place and time
  - Research: Generate easy-to-debug tests

What to Look For

- In simulation
  - Determinism
    - Does the failure always occur in the same settings?
      - With the same seed?
      - With different seeds (or random seed)?
  - Parameters that are correlated with the failure
    - Parameters that cause the failure to disappear
    - Parameters that cause the failure to change
  - Specific parts in the stimuli that are correlated to the failure
- In formal verification
  - Constraints that affect the failure
  - Time bounds that affect the failure

Resolving the Failure

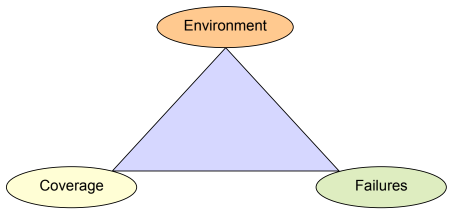
- This does not always mean fix the fault
  - Defer to future tape outs / releases
  - Bypass by software or surrounding modules
  - Record in errata sheets
- Need to ensure that the resolution is complete
  - The fix / bypass is correct
  - All cases are covered
  - No new faults introduced in the process
  - (Similar cases are also handled)
- Mini-verification plan is needed
  - Coverage models
  - Stimuli generation strategy
  - New result checkers

Adapting the Verification Plan and Process

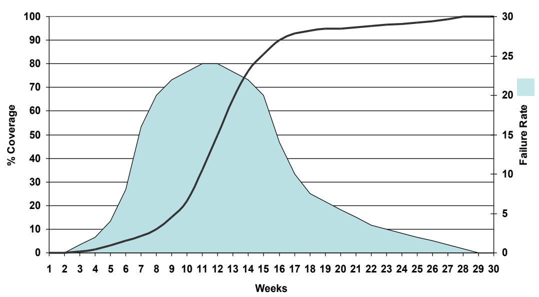
- Need to minimize faults found by chance or found too late
  - These faults can easily be missed if we are less lucky
- Indicators that faults are found by chance
  - Faults are not found at the right time
    - Fault is found at the wrong level of the hierarchy
    - Faults are found not at the area we concentrate on
    - Need to understand why faults are not found at the right time
      - And, change the plan and process accordingly
  - Faults are not found by the right checker
    - Only a side effect of the fault is detected
    - May indicate missing checker or problems in existing checker
  - Simulation with failure is not flagged by coverage
    - Does not activate uncovered or rarely covered coverage point
    - Indicates missing coverage models

Correlating Coverage and Failures

- There is a direct correlation between
  - Changes in the verification environment and the DUV
  - Progress in coverage
  - Detection of new failures



Correlating Failure Rate and Coverage Progress



Individual Coverage and Failure Correlation

- Correlating a failure to specific coverage can be helpful in the failure analysis and debugging processes
- Rare coverage points exercised by a simulation that fails can hint at the location of the fault that caused the failure
  - Rare coverage points are coverage points rarely, if ever, exercised by passing simulations
  - These coverage points record what happened in the DUV prior to the failure
  - They are very useful if the failure is distant (in logic or time) from the fault or the fault is complex
- If no such rare coverage points are recorded, then it is likely that the failure is found by chance
  - The verification plan needs to be refined to catch these failures

Regression

Skip to Escape Analysis

Regression Suites

- A regression suite is a set of tests that are run on the verified design on a regular basis
  - After major changes
  - Periodically: Every night or every weekend
- Regression goals
  - Assuring that things that worked did not stop working
    - This is vital because every bug fix, on average, introduces one fifth of a bug
  - Detecting "unexpected" bugs

Types of Regression

- Static regression
  - The regression suite is comprised of a set of "interesting" test patterns
    - Tests that found bugs in the past
    - Tests that reach corner cases
- Random regression
  - A.k.a. dynamic regression, probabilistic regression
  - The regression suite is comprised of a set of test specifications and an execution policy
    - For example, execute 100 tests of specification A, 35 tests of specification B, and 20 tests of specification C

Static Vs. Random Regression

- Static regression
  - ☑ Known, guaranteed quality
  - ☒ Sensitive to changes
  - ☒ Hard to maintain
- Random regression
  - ☒ Unknown quality
  - ☑ Less sensitive to changes
  - ☑ Easy to maintain
  - ☑ Easy to adapt to simulation resources

The Preferred Solution

- Combination of static and random suites
- Small static suite with hard to recreate cases
  - Hard to reach corner cases
  - Tests that discovered hard to find bugs
- Random suites for everything else

Regression Suites Requirements

- A regression suite must be:
  - **Comprehensive** so that it is likely to catch all the bugs introduced
  - **Small** so that it can economically be executed many times
- **How can we make our regression suite small and comprehensive?**
- **Solution:** use coverage information
  - Select a set of tests that achieve 100% coverage (of the coverage achieved so far)
  - Select the smallest possible such set

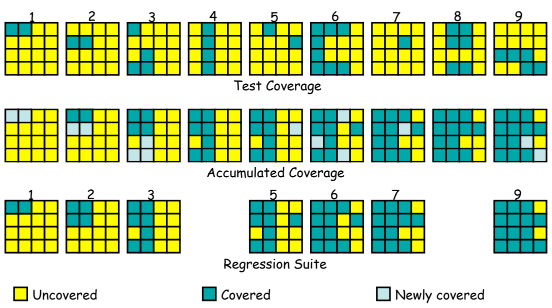
The Set Cover Problem

- Let S = {C_1,...,C_n} be the set of coverage tasks
- Let T = {T_1,...,T_m} be a set of tests
  - Each test T_i covers subset {C_{i1}, C_{i2}, ...} of the coverage tasks
- The **set cover problem:** Find the smallest subset of T that covers S
- The set cover problem is a known NP-Complete problem
  - However, there are a number of good algorithms for it

Online Algorithm

- For each new test T
  - If T covers an uncovered coverage task
    - Add T to the regression suite
- Advantages
  - Very simple
  - Low memory requirements

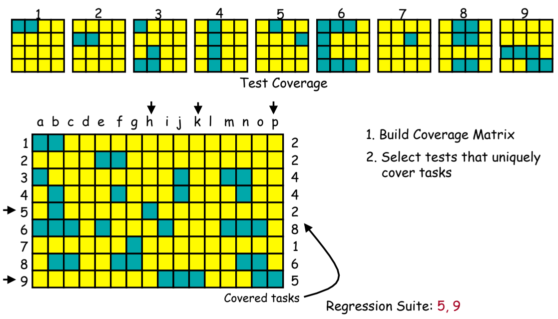
Online Algorithm Example



Greedy Algorithm

- Initialization
  - Build coverage matrix tests vs. tasks
  - Select tests that uniquely cover tasks
- Loop until all covered tasks are removed
  - Remove all the tasks covered by selected tests
  - Choose the test that covers most remaining tasks
- Advantages
  - Complexity is polynomial in the number of tests and coverage tasks
  - Quality solution
- Disadvantage
  - Requires to keep the entire coverage matrix in memory

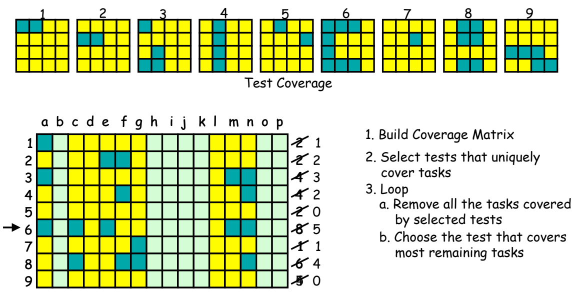
Greedy Algorithm Example



1. Build Coverage Matrix
2. Select tests that uniquely cover tasks

Regression Suite: 5, 9

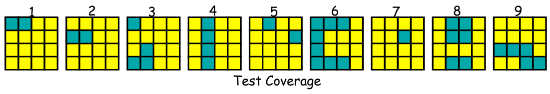
Greedy Algorithm Example

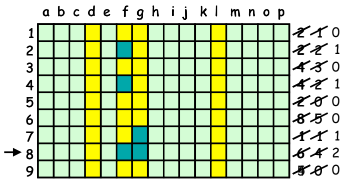


1. Build Coverage Matrix
2. Select tests that uniquely cover tasks
3. Loop
   a. Remove all the tasks covered by selected tests
   b. Choose the test that covers most remaining tasks

Regression Suite: 5, 9, 6

Greedy Algorithm Example

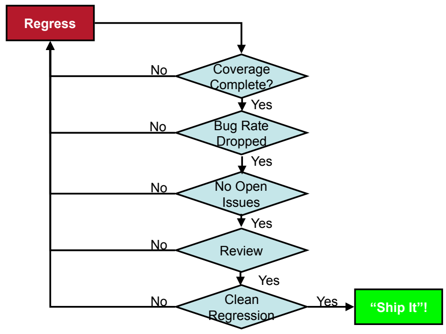




1. Build Coverage Matrix
2. Select tests that uniquely cover tasks
3. Loop
   a. Remove all the tasks covered by selected tests
   b. Choose the test that covers most remaining tasks

Regression Suite: 5, 9, 6, 8

When Is Verification Done?



Tape-Out Readiness

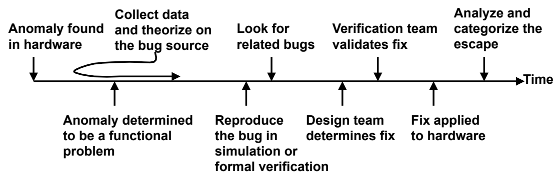
- Before sending a design to manufacturing, it must meet established tape-out criteria
- The criteria is a series of checklists that indicate completion of planned work
- Verification is just one element in this series of checklists
- Tape-out readiness is measured by a set of metrics
- The most relevant metrics for verification are bug rates and coverage

Escape Analysis

Escape Analysis

- An **escape** is a bug found later in the verification process than it should have been
  - In other words, it escaped its target place
  - Usually, escapes refer to bugs found in the hardware itself instead of during simulation
- Escape analysis has two important aspects
  - Make sure that the bug is fully understood and fixed correctly
    - We do not want another tape-out because of a bad fix
  - Understand why the bug escaped simulation in the first place and try to improve the verification plan and process to avoid such escapes in the future

Individual Escape Analysis Timeline



How big is Exhaustive?

- Consider simulating a typical CPU design
  - 500k gates, 20k DFFs, 500 inputs
  - 70 billion sim cycles, running on 200 linux boxes for a week
  - **How big: 2^{36} cycles**
- Consider formally verifying this design
  - Input sequences: cycles $2^{(inputs+state)} = 2^{20500}$
  - What about X's: 2^{15000} (5,000 X-assignments + 10,000 non-reset DFFs)
  - **How big: 2^{20500} cycles** (2^{15000} combinations of X is not significant here!)
- That's a big number!

– Cycles to simulate the 500k design:	$\mathbf{2^{36}}$	(70 billion)
– Cycles to formally verify a 32-bit adder:	2^{64}	(18 billion billion)
– Number of stars in universe:	2^{70}	(10^{21})
– Number of atoms in the universe:	2^{260}	(10^{78})
– Possible X combinations in 500k design:	2^{15000}	(10^{4515} x 3)
– Cycles to formally verify the 500k design:	$\mathbf{2^{20500}}$	(10^{6171})

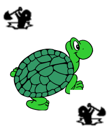
49

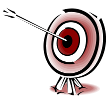
Summary

- Completion of the Verification Cycle includes:
  - Coverage analysis
  - Failure analysis
  - Regression
  - Tape-out readiness
  - Escape analysis

50

Are We There Yet?





51