Verification of Digital Systems

Unit 1 : Introduction Part 1

Vinay Reddy

Department of Electronics & Communication Engineering

Course Objectives:

- **Understand and use the SystemVerilog** RTL design and synthesis features, including new data types, literals, procedural blocks, statements, and operators, relaxation of Verilog language rules, fixes for synthesis issues, enhancements to tasks and functions, new hierarchy and connectivity features, and interfaces.

- **Appreciate and apply the SystemVerilog verification features**, including classes, constrained random stimulus, coverage, strings, queues and dynamic arrays, and learn how to utilize these features for more effective and efficient verification.

Course Outcomes:

- **Students will be able to write systemverilog code for both design and verification.**

- **Students using System Verilog constructs will be able to achieve 100% (a good) functional coverage.**

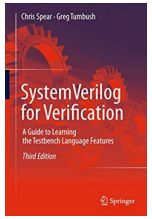
Syllabus

Unit 1 – Introduction Part 1

1	Verification Guidelines: The Verification Process	1.1	Chapter 1
2	Basic Test bench Functionality	1.3	
3	Directed Testing	1.4	
4	Methodology Basics	1.5	
5	Constrained-Random Stimulus	1.6	

Text Book

"**SystemVerilog for Verification**: A Guide to Learning the Testbench Language Features", **Chris Spear**, Greg Tumbush, Springer Publication, ISBN-13: 9781489995001 , 2014



https://3ec1218usm.files.wordpress.com/2016/12/book_systemverilog_for_verification.pdf

What is the goal of verification?

What is the goal of verification?

If you answered, "**Finding bugs**," you are only partly correct.

Unit 1 : Introduction

1. The Verification Process

Goal of hardware design

To create a device that performs a particular task, such as a DVD player, network router, or radar signal processor, based on a design specification.

Purpose as a verification engineer

To make sure the device can accomplish that task successfully

The design is an accurate representation of the specification.

The behavior of the device, when used outside of its original purpose, is not your responsibility, although you want to know where those boundaries lie.

1. The Verification Process

The process of verification parallels the design creation process.

Designers Side

Designer reads the hardware specification for a block.

Interprets the human language description.

creates the corresponding logic in a machine-readable form, usually RTL code.

To do this, the designer needs to understand the **input format**, **the transformation function**, and the **format of the output.**

Verification Side

There is always ambiguity in this interpretation, perhaps because of ambiguities in the original document, missing details, or conflicting descriptions.

Verification engineer, you must also read the hardware specification.

Create the verification plan.

Follow it to build tests showing the RTL code correctly implements the features.

BLOCK LEVEL

Testing at Different Levels

The easiest ones to detect are at the **block level**

Examples –

ALU Block — Did the ALU correctly add two numbers?

Arbiter — Did every bus transaction successfully complete?

Router — Did all the packets make it through a portion of a network switch?

INTEGRATION LEVEL

Testing at Different Levels

Integration phase

- ✔ After the block level, the next place to look for discrepancies is at boundaries between blocks.
- ✔ For a given protocol, what signals change and when?

PES UNIVERSITY ONLINE

1. The Verification Process

Testing at Different Levels

To simulate a single design block, you need to create tests that generate stimuli from all the surrounding blocks — a difficult chore.

As you start to integrate design blocks, they can stimulate each other, reducing your workload.

These multiple block simulations may uncover more bugs, but they also run slower.

SYSTEM LEVEL

Testing at Different Levels

Note: the simulation performance is greatly reduced

- ✔ **At the highest level of the Design Under Test (DUT), the entire system is tested.**

- ✔ **Your tests should strive to have**
  - All blocks performing interesting activities concurrently.
  - All I/O ports are active,
  - Processors are crunching data, and
  - Caches are being refilled.

- ✔ **With all this action, data alignment and timing bugs are sure to occur.**

ERROR HANDLING

Testing at Different Levels

- ✔ Once you have verified that the DUT performs its designated functions correctly, you need to see how it operates when there are errors.

- ✔ Can the design handle a partial transaction, or one with corrupted data or control fields?

Error injection and handling can be the most challenging part of verification.

1. The Verification Process

✔ **At the block level,**

you can show that individual cells flow through the blocks of an ATM router correctly.

✔ **but at the system level,**

you might have to consider what happens if there are streams of different priority.

Which cell should be chosen next is not always obvious at the highest level.

you can never prove there are no bugs left, so you need to constantly come up with new verification tactics.

The Verification Plan

The verification plan is derived from the hardware specification and contains a description of what features need to be exercised and the techniques to be used.

These steps may include

- directed or random testing,
- assertions,
- HW/SW co-verification,
- emulation, or
- use of verification IP.

PES UNIVERSITY ONLINE

1. The Verification Process

The Verification Plan

TASK 1 :

Students can form a group of two –

Student 1: Play the role of Design Engineer
Student 2: Play the role of Verification Engineer

Task : Choose a specification for an ALU design, both the design and verification engineer should write the code in parallel
Design engineer – Concentrate on the Design code
Verification engineer – Concentrate on Testbench code and verification plan

Perform functional verification using the TOOL of your choice –
Xilinx Vivado
Intel Quatus
EDA Playground

How will you make sure you code is 100% functionally correct?
Is there any metric you have which can prove that your code is 100% functionally correct?

Unit 1 : Introduction

2. The Verification Methodology Manual

The *Verification Methodology Manual for SystemVerilog* is a blueprint for system-on-chip (SoC) verification success.

To develop testbenches in the earlier 2000 we used - **vera** and **e** Verification language

RVM and **eRM** methodologies were used respectively

Today we use SystemVerilog – Popularly used verification language.

SystemVerilog – HDVL (Hardware Design and Verification Language)

It has gained popularity because it is easy to develop maintainable & reusable verification environment.

Methodologies for SV –
OVM (Open Verification Methodology) - Cadence
AVM (Advanced Verification Methodology) - Mentor Graphics
VMM (Verification Methodology Manual) - Synopsis
UVM (Universal Verification Methodology) – Accellera Company

3. Basic Testbench Functionality

The purpose of a testbench is to determine the correctness of the DUT.

This is accomplished by the following steps.

- **Generate stimulus**
- **Apply stimulus to the DUT**
- **Capture the response**
- **Check for correctness**
- **Measure progress against the overall verification goals**

4. Directed Testing

Directed Testing approach:

You look at the hardware specification and write a verification plan with a **list of tests**, each of which concentrated on a set of related features.

Advantage:

Produces almost immediate results, since little infrastructure is needed when you are guiding the creation of every stimulus vector.

Given ample time and staffing, directed testing is sufficient to verify many designs.

Disadvantage:

When the design complexity doubles, it takes twice as long to complete or requires twice as many people to implement it.

4. Directed Testing

Directed Testing Procedure:

You write stimulus vectors that exercise the features in the DUT.

You then simulate the DUT with these vectors and manually review the resulting log files and waveforms to make sure the design does what you expect.

Once the test works correctly, you check it off in the verification plan and move to the next one.

This incremental approach makes steady progress.

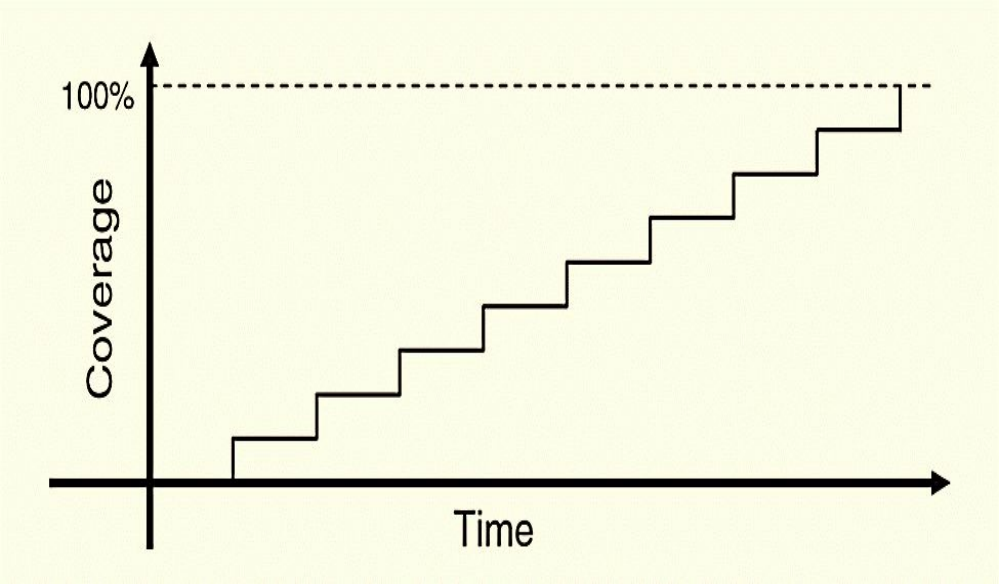


Fig: Directed test progress over time

4. Directed Testing

In this design space there are many features, some of which have bugs.

You need to write tests that cover all the features and find the bugs.

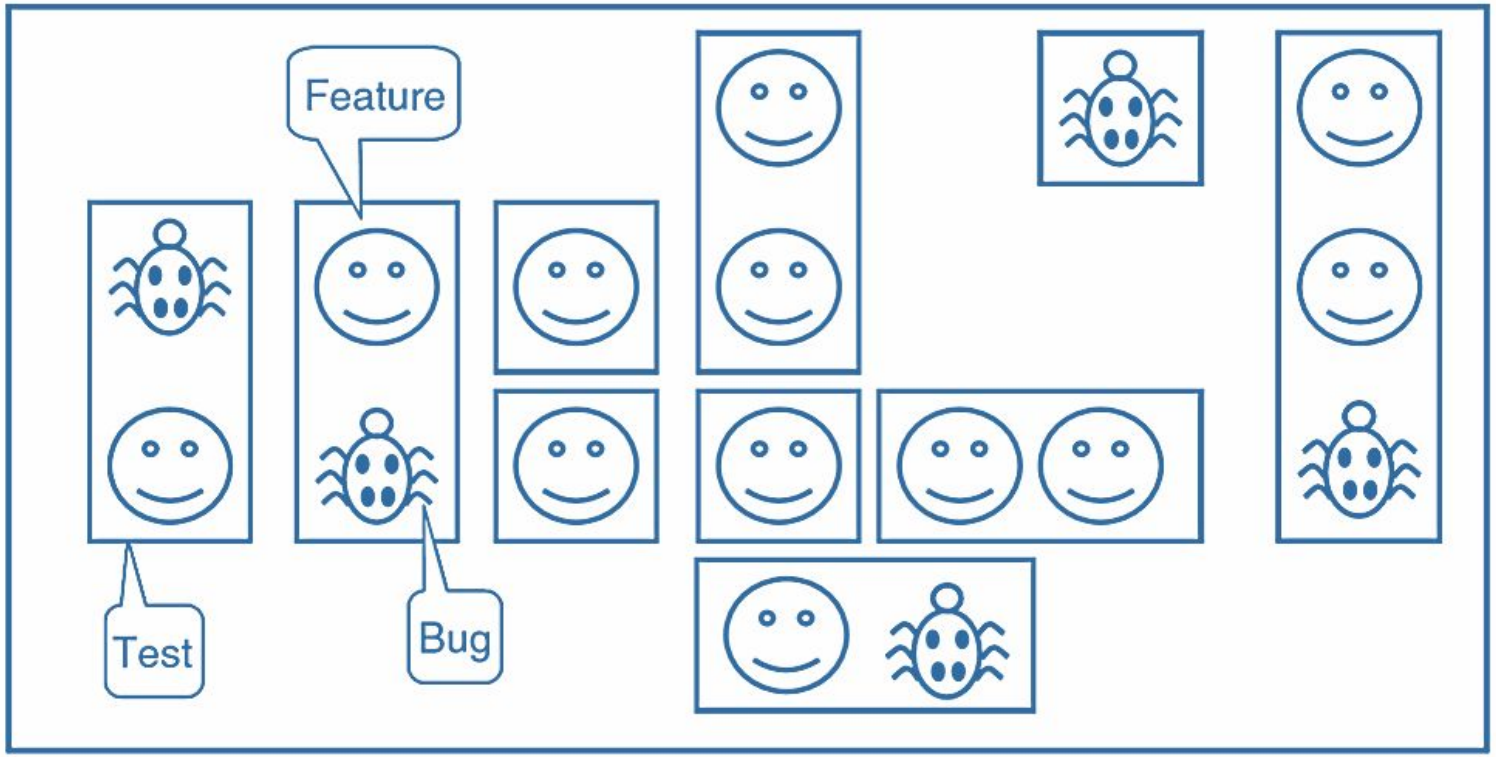


Fig: Directed test coverage

Unit 1 : Introduction

4. Directed Testing

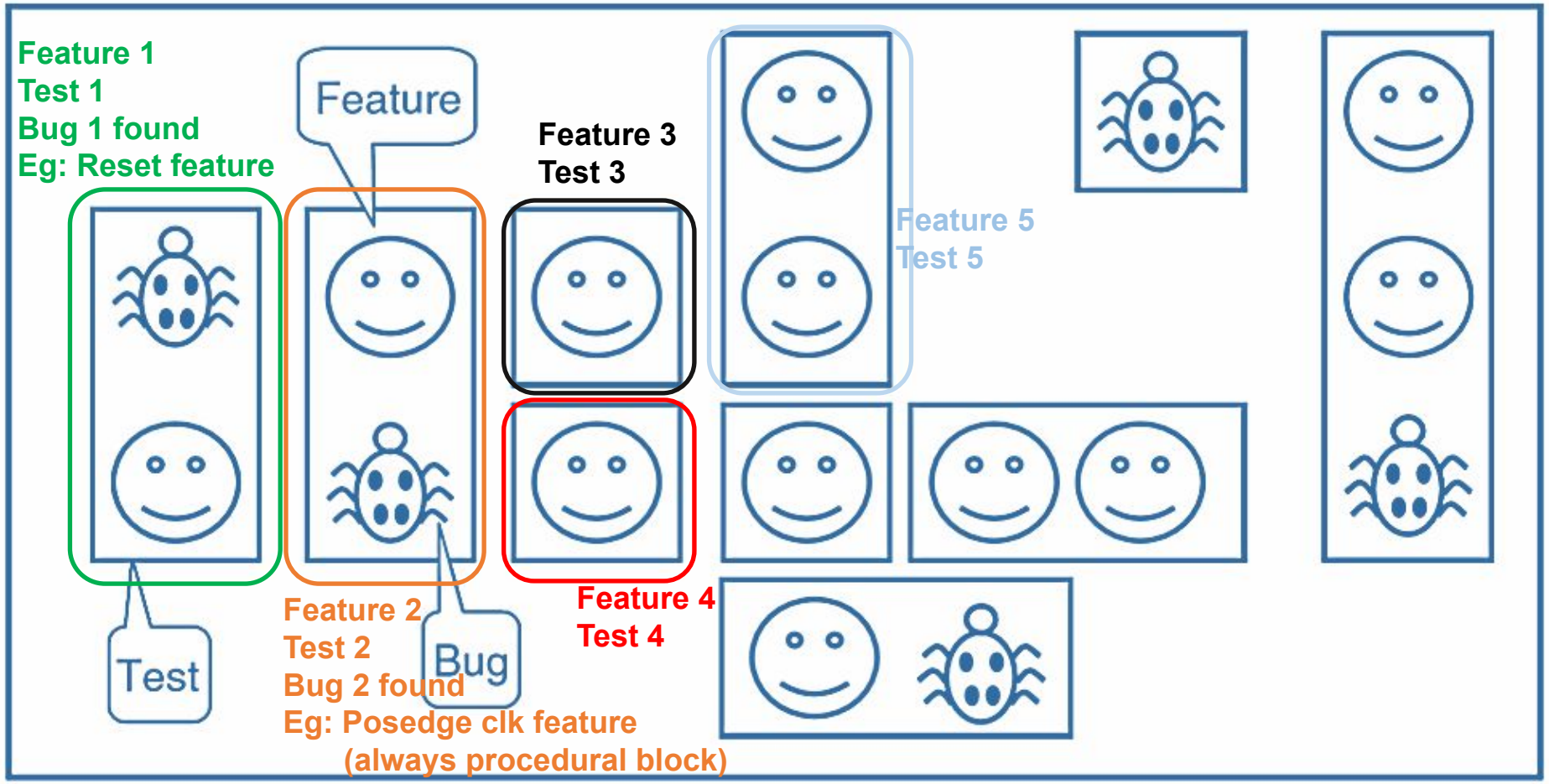
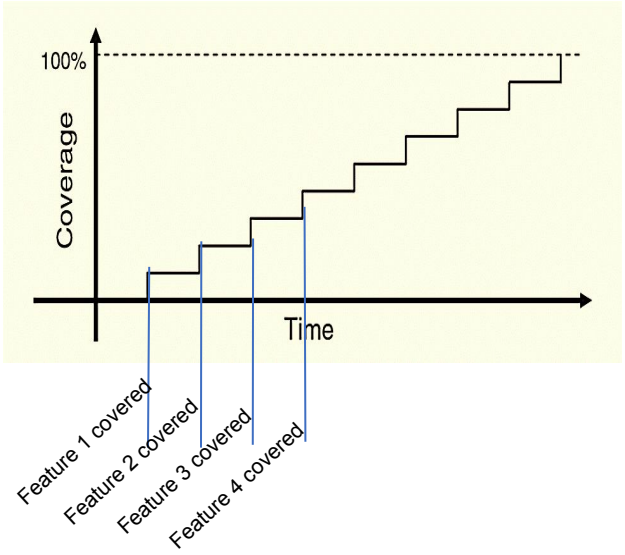


Fig: Directed test coverage



Once all the features are tested, the coverage reaches 100%

5. Methodology Basics

This book uses the following principles for verification

- Constrained-random stimulus
- Functional coverage
- Layered testbench using transactors
- Common testbench for all tests
- Test case-specific code kept separate from testbench

Constrained-random stimulus:

Random stimulus is crucial for exercising complex designs instead of applying directed stimulus

Functional coverage:

When using random stimuli, you need functional coverage to measure verification progress.

Layered testbench using transactors:

Furthermore, once you start using automatically generated stimuli, you need an automated way to predict the results — generally a scoreboard or reference model. Building the testbench infrastructure, including self-prediction, takes a significant amount of work.

A layered testbench helps you control the complexity by breaking the problem into manageable pieces.

Common testbench for all tests:

With appropriate planning, you can build a testbench infrastructure that can be shared by all tests and does not have to be continually modified.

Test case-specific code kept separate from testbench:

Code specific to a single test must be kept separate from the testbench to prevent it from complicating the infrastructure.

5. Methodology Basics

Building this style of testbench takes longer than a traditional directed testbench — especially the self-checking portions.

As a result, there may be a significant delay before the first test can be run.

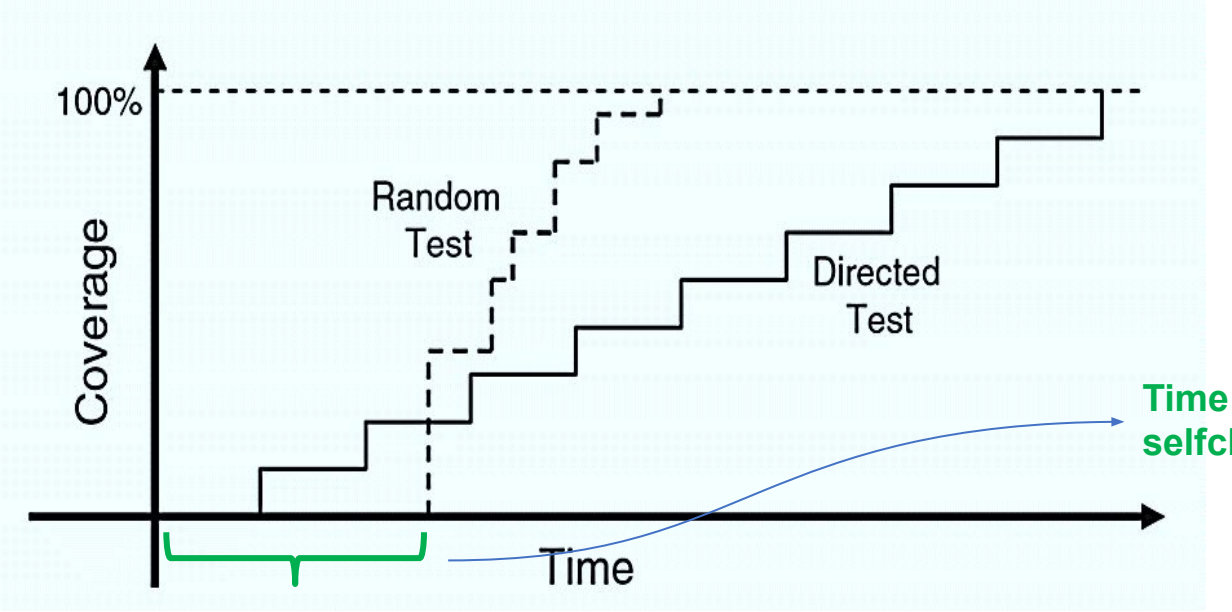


Fig: Constrained-random test progress over time vs. directed testing

5. Methodology Basics

- ✔ Every random test you create shares this common testbench, as opposed to directed tests where each is written from scratch.

- ✔ The result is that your single constrained-random testbench is now finding bugs faster than the many directed ones.

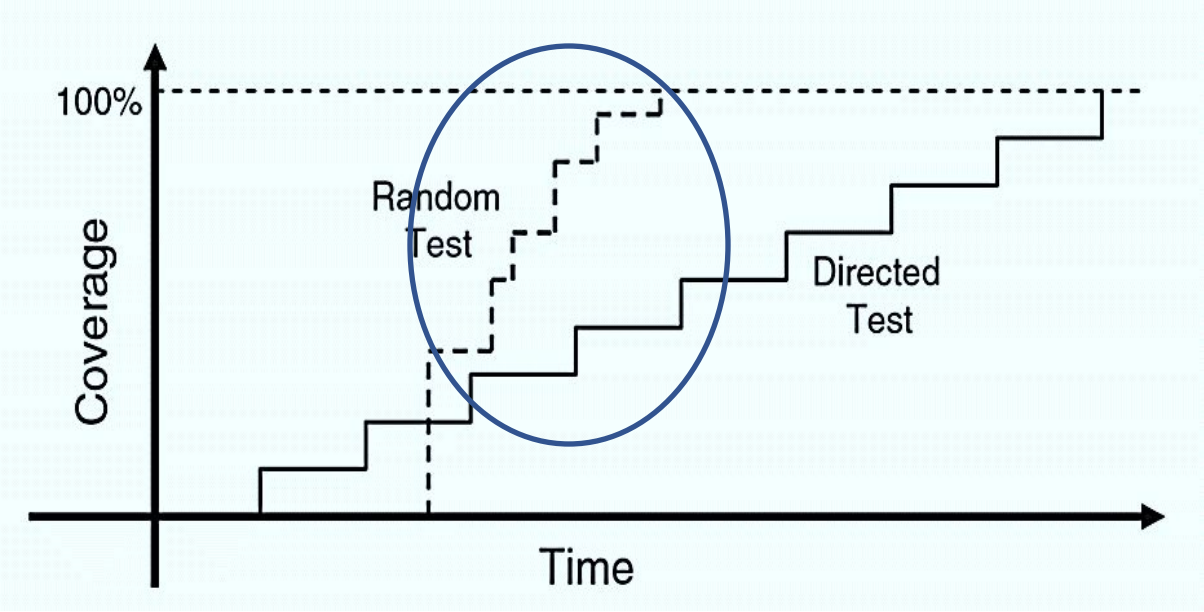


Fig: Constrained-random test progress over time vs. directed testing

6. Constrained-Random Stimulus

- ✔ Although you want the simulator to generate the stimulus, you don’t want totally random values.
- ✔ Constraining the random values to become relevant stimuli.
- ✔ These values are sent into the design, and are also sent into a high-level model that predicts what the result should be.
- ✔ The design’s actual output is compared with the predicted output.

6. Constrained-Random Stimulus

Coverage for constrained-random tests over the total design space

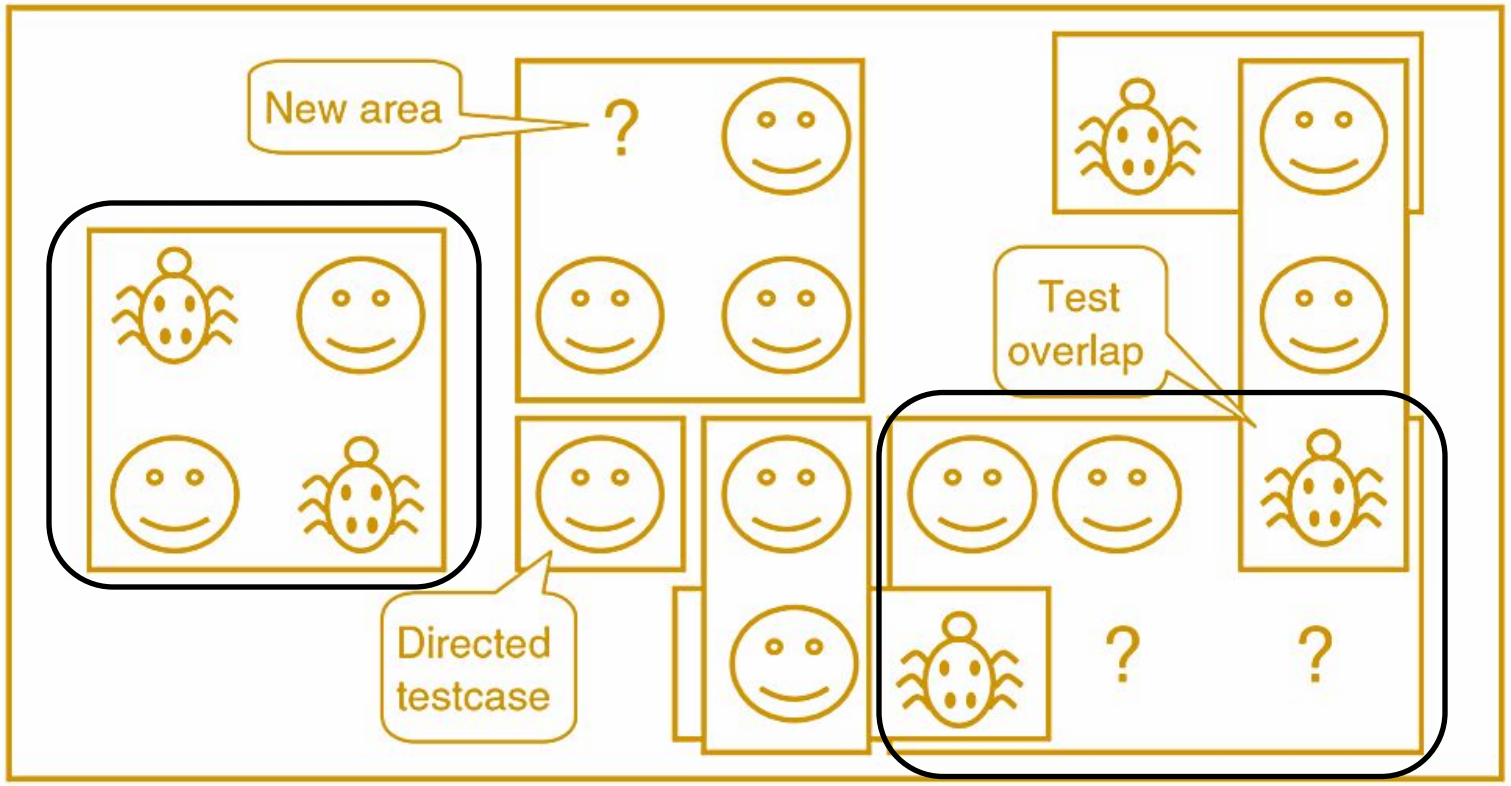


Fig: Constrained-random test coverage

Random test often covers a wider space than a directed one.

PES UNIVERSITY ONLINE

6. Constrained-Random Stimulus

Coverage for constrained-random tests over the total design space

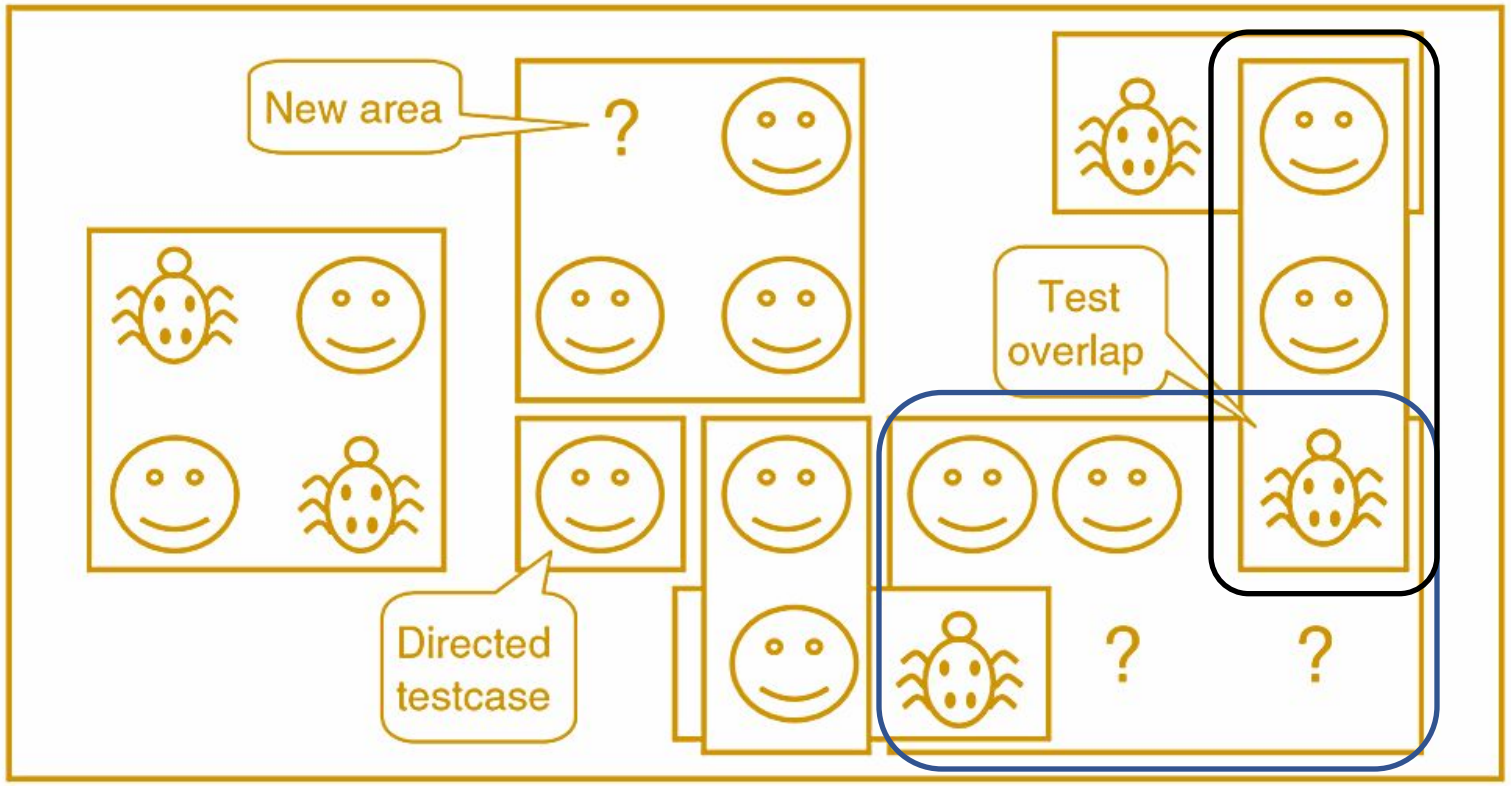


Fig: Constrained-random test coverage

This extra coverage may overlap other tests, or may explore new areas that you did not anticipate.

If these new areas find a bug, you are in luck!

6. Constrained-Random Stimulus

Coverage for constrained-random tests over the total design space

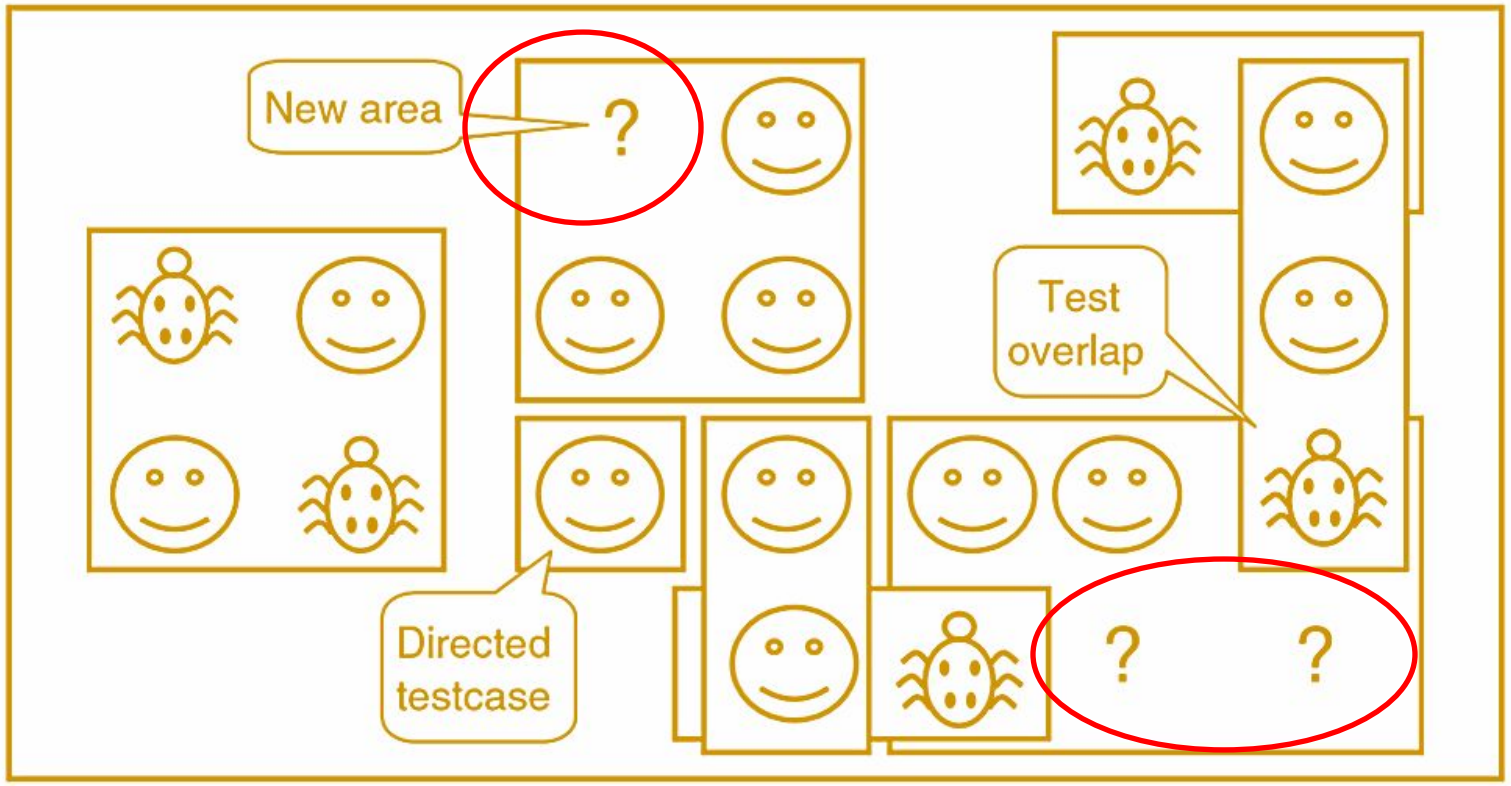


Fig: Constrained-random test coverage

write more constraints to keep random generation from creating illegal design

6. Constrained-Random Stimulus

Coverage for constrained-random tests over the total design space

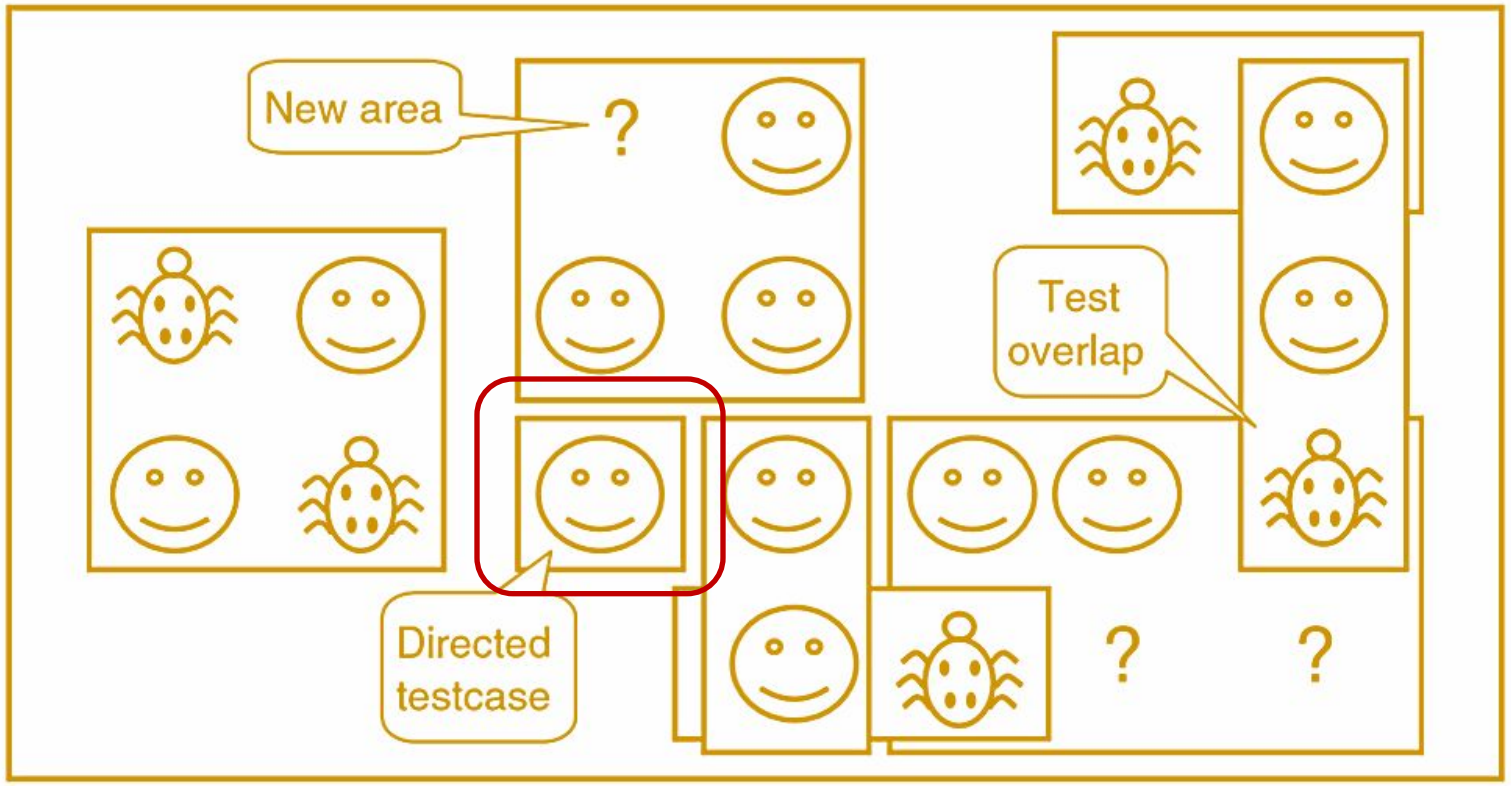


Fig: Constrained-random test coverage

Lastly, you may still have to write a few directed tests to find cases not covered by any other constrained-random tests.

Note: If you create a random testbench, you can always constrain it to created directed tests, but a directed testbench can never be turned into a true random testbench.

6. Constrained-Random Stimulus

Paths to achieve complete coverage

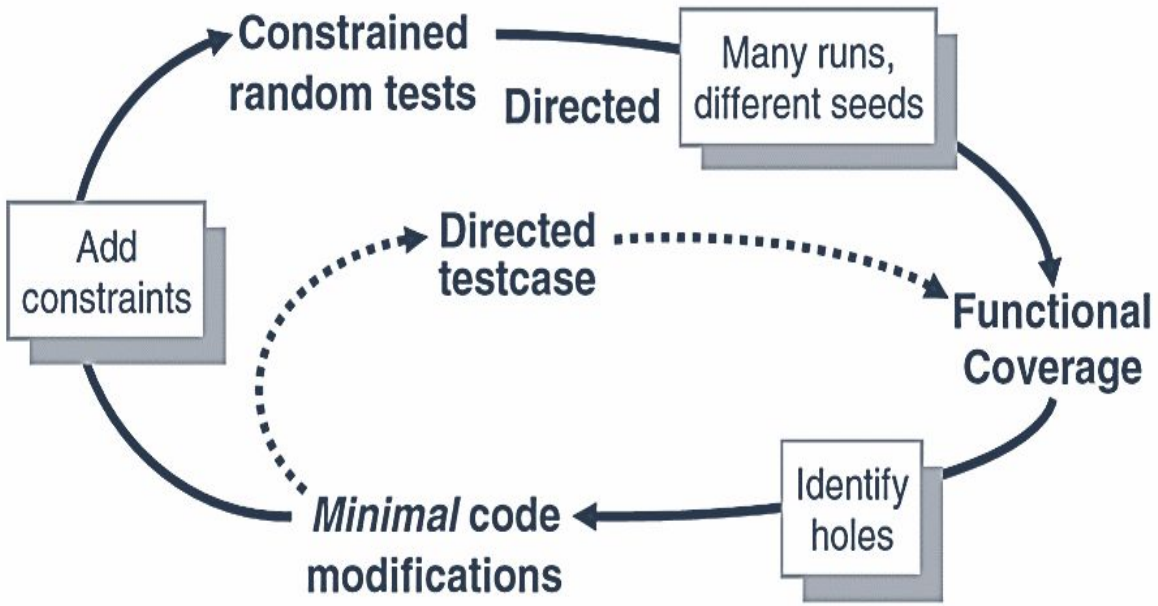


Fig: Coverage convergence

Start at the upper left with basic constrained-random tests. Run them with many different seeds.

When you look at the functional coverage reports, find the holes where there are gaps in the coverage.

Then you make minimal code changes, perhaps by using new constraints.

Spend most of your time in this outer loop, writing directed tests for only the few features that are very unlikely to be reached by random tests.

Reference

"**SystemVerilog for Verification**: A Guide to Learning the Testbench Language Features", **Chris Spear**, Greg Tumbush, Springer Publication, ISBN-13: 9781489995001 , 2014.

Text book pg numbers: 1 – 9

Chapter: 1

Topics: 1.1,1.3,1.4,1.5,1.6

Extra Reading Recommendations:

Youtube video reference for the topic **Directed Vs Random verification**

-https://www.youtube.com/watch?v=fkyu88hMwwo&t=334s

Thank You

Vinay Reddy

Department of Electronics & Communication Engineering