Verification Cycle, Verification Methodology & Verification Plan

Functional specification -> Designer implements the functional specification (in HDL) -> create verification plan -> develop verification environment -> debugging -> run regression -> tape out readiness -> debug fabricated hardware -> escape analysis

Create verification Plan

- functions to be verified
- resources required and schedule details
- required tools
- specific tests and methods
- completion criteria - how do you measure success?

Develop verification environment

- set of software code and tools that enable the verification engineer to identify flaws in the design
- stimulus and checking for stimulation based environments
- rules generation (properties) for formal verification environments

Debug HDL and environment

- run tests according to the verification plan and look for anomalies
- examine the anomalies to reveal failure source
- fix cause of the failure
- rerun the same test
- update verification plan based on the test results

Run regression

- continuous running of the tests defined in the verification plan
- simulation farms
- used to uncover hard-to-find bugs and ensure that the quality of the design keeps improving

Debug fabricated hardware

- hardware bring-up

Perform escape analysis

- reproduce the bug in a simulation environment, if possible
- fully understand the bug -> reasons why it went undiscovered by the verification environments

Verification methodology

Test patterns

- patterns created to test specific behaviours
- each pattern handles a single scenario
- hand generated
- DUV behaviour is manually checked
- bare expensive

Test cases

- mostly change the checking of the test
- checking evolves to self checking tests/automatic checking (checking is independent of the stimulus)

Test patterns -> test cases -> test generators -> test drivers

Test case generators

- machine generated random patterns
- more generic targets, multiple scenarios and large number of tests

Test case drivers

- stimuli generation is embedded in the verification environment
- stimuli are generated the operation of the environment (and stimulation)
- driver can react to the state of the DUV

Evolution of the verification Plan

- functional specification document
- understand the DUV before determining how to verify it
- confront unclear and ambiguous definitions
- incomplete and changing continuously

Verification Plan

- verification levels – which levels to perform the verification – complexity, resources, risk, existence of a clean interface and spec
- functions – specific functions of the DUV that the verification team will exercise – assign priority for each function — critical functions — secondary functions – functions not verified at this level — fully verified at a lower level — not applicable to this level – required tools – specific tests and methods — type of verification: white, black, grey — verification strategy: formal verification, deterministic, random based — abstraction level: from bit-level to algorithmic level from bit-streams to transactions — checking: simple IO checking for data correctness, behavioural rules for timing – abstraction levels — bit-level -> command & data "packet" level -> sequence level -> program or algorithmic level – coverage requirements — feedback mechanism that evaluates the quality of the stimuli — required in all random-based verification environments — defined as events – completion criteria