SpringSoft
Accelerating Engineers

# Certitude™ Close the "Quality Gap" in Functional Verification

*Reading November 2011*

## Topics

- Why Verification
- Existing tools and techniques
- Introducing Functional Qualification
- Certitude™ Functional Qualification System
- Q&A

## Why verification?

- Designs are made by humans
- Humans have
  - Finite / limited capacity
    - In envisioning consequences of decisions
    - Memory
    - Attention
  - Make mistakes
    - Sometimes very creative/convoluted/surprising ones
    - Sometimes very gross and visible
      - But may still go unnoticed for long
  - Hence there are bugs in designs
- However customers / consumers want
  - Bug free designs
  - Safer, more reliable objects
  - More features, more powerful objects

=> Have computers make designs
=> Wait for bugs to be reported by the users
=> Audit the design to squeeze out the bugs

## Verification

- Techniques to find bugs in designs
  - Assisted by computers
    - For tests generation
    - To check simulation results
  - The grand scheme is from humans

- But there is a little hurdle…

# Design State Space

- Number of states for a design
  - $2^{FF} \sim 10^{0.7*FF}$
    - FF: number of flip-flops
  - Some states are unreachable
- Estimated number of FF in a modern design
  - > 100 k
- Estimated number of stars in the universe
  - $10^{23}$
- Estimated number of atoms in the universe
  - $10^{80}$
- Estimated age of the universe, in seconds
  - $4*10^{17}$
- CPU clock frequency
  - 4 GHz => $4*10^{9}$

SpringSoft
Accelerating Engineers

## Verification

- Verification issues
  - Brute force / exhaustiveness is out of reach
- Consequences
  - Verification is partial
    - Art: what to verify, at which level, methods
    - Risk management: what to skip, when to stop
  - Subject to bugs
    - Missing / broken checkers
    - Missing tests
    - Missing scenario
  - Bugs in the VE hide bugs in the design

⇒ Audit the verification to squeeze out VE and design bugs

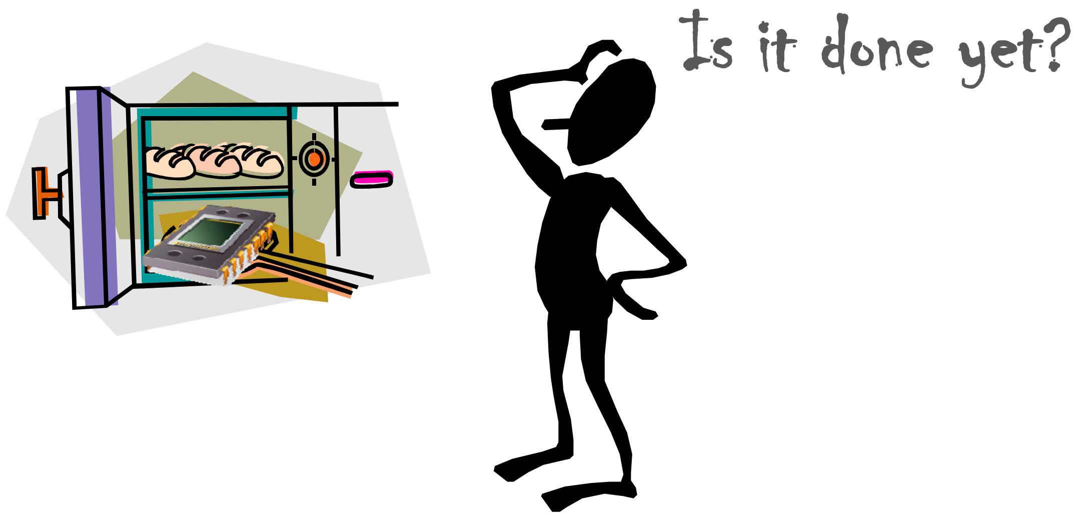
## Verification audit

- Traditionally
  - Human only based process
  - No clues about
    - Where to search
    - What to search
    - If there is anything to find
- Computer assisted process

  => Write a VE for the VE...

  => Use Certitude
    - Generates broken designs to check the VE

## Verification 1M$ Question



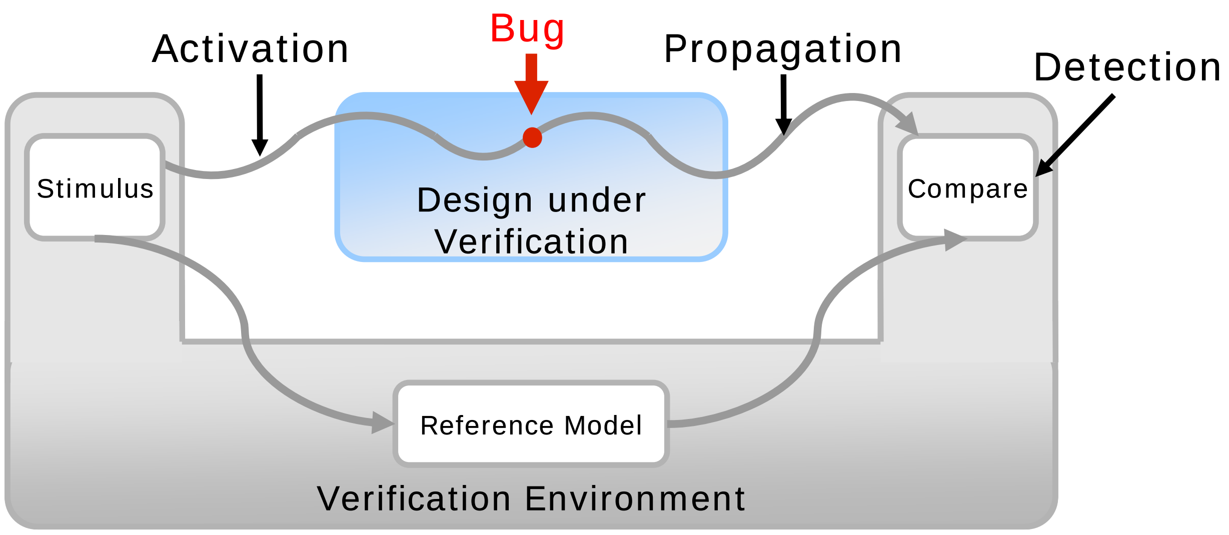
- Does my stimulus *cover* all scenarios effectively?
- Am I *monitoring and checking* all important functionality per the spec?
- Are *infrastructure or process issues* masking serious RTL bugs?

## Impact of Poor Verification Environment

- Wasted simulation cycles
  - Missing / buggy checkers and assertions mean RTL bugs go undetected even with more tests and CPU time
- Inefficient resource allocation
  - Which blocks are poorly verified?
  - Do I need better stimulus / scenario?
- Late-stage identification of problems
  - ECOs required close to tape-out
- Silicon re-spins
  - Bug escapes → Significant $$

## Effective Verification

*It's all about activation, propagation, and detection*

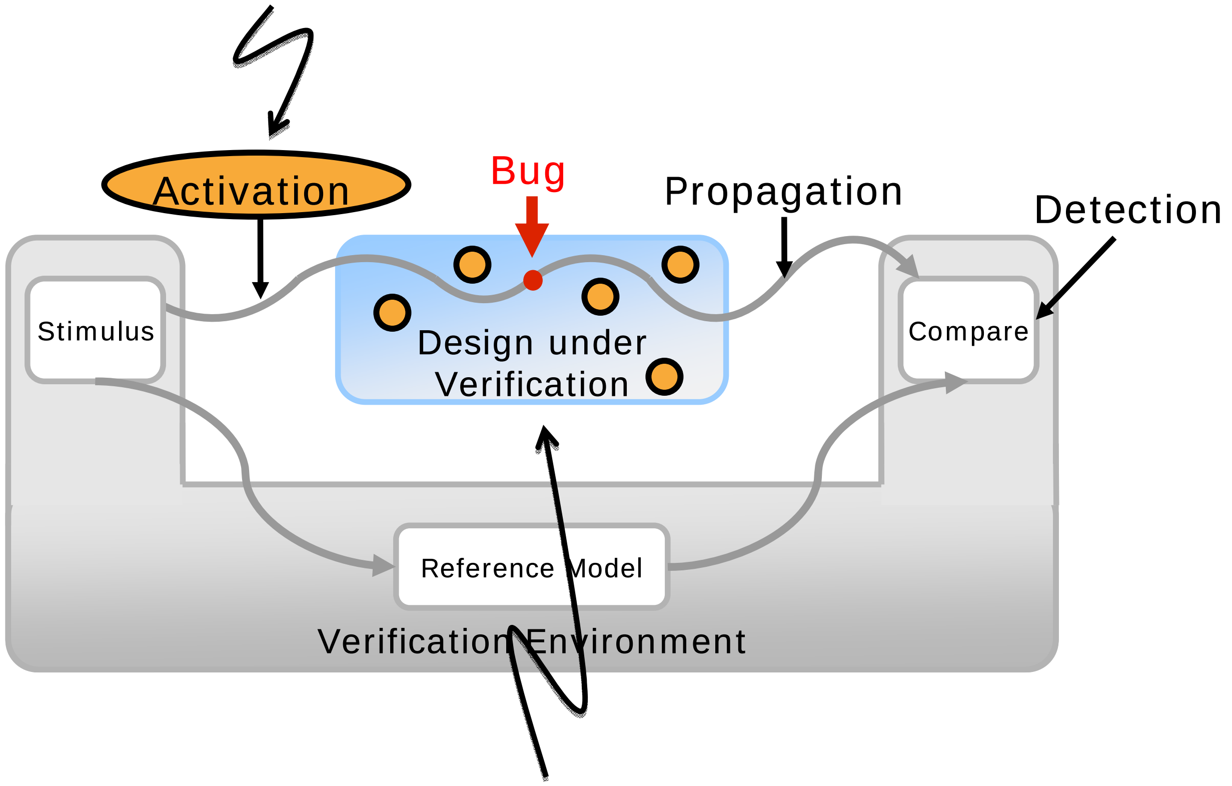


*To detect a bug…*

- The stimulus must **activate** the buggy logic
- An effect of the bug must **propagate** to an observation point
- The environment must **detect** the behavior difference due to the bug

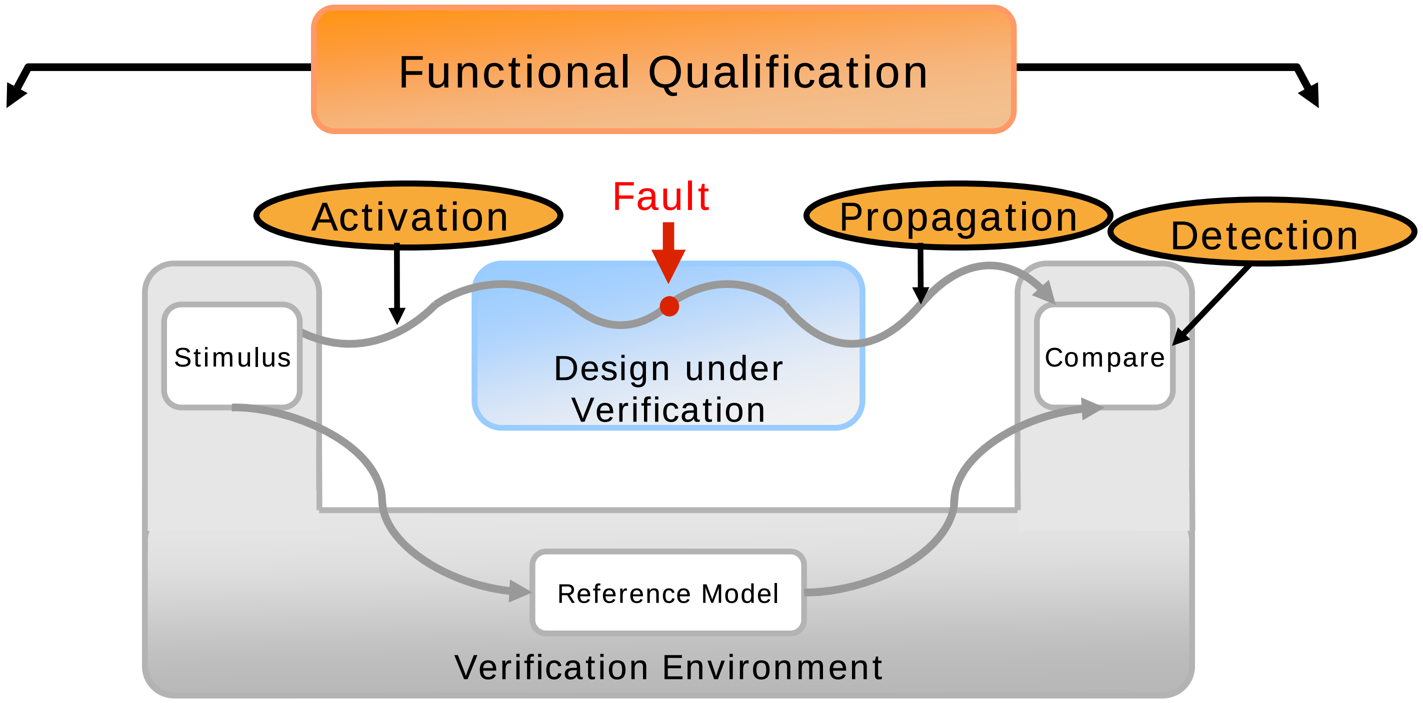
## Existing Tools are Insufficient

***Code coverage*** *measures activation, but says nothing about propagation or detection*



***Functional coverage*** *checks "important" functional points, but is subjective and incomplete*

## Introducing Functional Qualification



- Inserts "artificial bugs" called faults into the design
- Measures the ability of the verification environment to **activate**, **propagate**, and **detect** the faults
- "Qualification" of the verification environment against many inserted faults provides objective measure of overall quality and identifies holes and weaknesses

## How it Works

- Modifies RTL code to insert faults

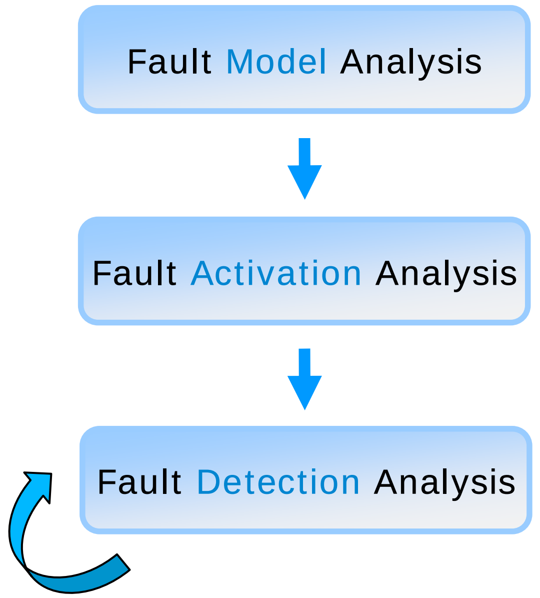
```
a = b | c   →  a = b & c  // change operator

if (a)      →  if (TRUE)  // force execution of "if" branch
  f1();          f1();
else           ~~else~~
  f2();          ~~f2();~~
```

- Simulates the broken RTL code with your third-party simulator and test suite
  - Does at least one test fail? *Great!*
    - Your environment is robust enough to detect that the RTL is broken
  - Do all tests pass? *Help!*
    - You now have two versions of the RTL, both of which are compliant with your verification environment

## Interpreting and Using the Results

- Results provide an *objective* measure of verification environment quality
  - Unbiased assessment of all three areas: Activation, propagation, and detection
- Undetected faults point to holes or weaknesses in the verification environment
  - Correlate to potential “real bugs” that have a similar effect…and would be missed by the environment
- Fixing these problems makes the verification environment more robust
  - Plugs holes that can allow bugs to slip through the process undetected

# Process & Flow



Analyze the design (static) to determine the possible faults

- Write instrumented RTL
- ID undetectable faults

Analyze the verification environment by simulating tests once

- ID non-activated faults
- Correlate faults and tests

Measure the ability to detect bugs by injecting faults and simulating

- ID non-detected faults
- Report details to direct fix

Fix and iterate as problems are found

## Methodology & Technology

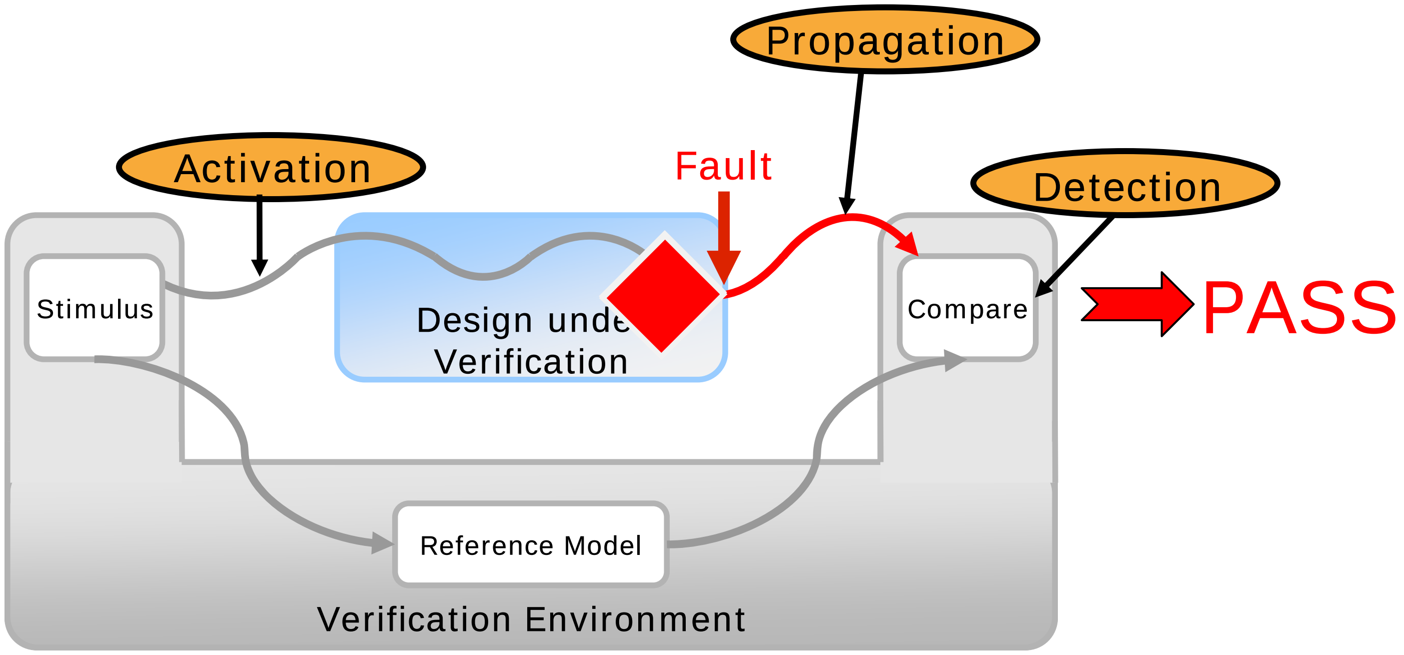
*Keys to Practical Application of Certitude*

- Find and fix important verification environment weaknesses…
  - Quickly
  - With minimal simulation time
  - With minimal engineer debug/analysis effort
- How? *Methodology*
  - Run incrementally
  - Run Certitude with a subset of tests
- How? *Technology*
  - Fault classes and priority
  - Fault dropping

## Methodology & Technology

- Some faults are more important than others
  - More likely to find big weaknesses
  - Easier to analyze
  - Fix likely to correct other problems
- Use only a few tests/seeds, especially at the beginning
  - An ND fault has already be activated and propagated to an output
    - => No problem with stimuli
    - => Problem with checkers

## Methodology & Technology



- A big design change is not reported by the VE
  => There is a serious verification issue here!

## Certitude™ Functional Qualification System

- The industry's first and only Functional Qualification solution
- Production-proven
  - Extensive worldwide customer list
  - Processor, communication, automotive, network, storage, wireless, multimedia, …
- Supports all common RTL languages and simulators
  - Verilog, VHDL, SystemVerilog
  - VCS™, Questa™, IUS™/NC™
- Supports all verification environments
  - Verilog, VHDL, SystemVerilog, C/C++, Specman/e, Vera, …
- Integrates quickly and easily with your simulation environment
  - Based on infrastructure already available in your regression flow

SpringSoft
Accelerating Engineers

# Benefits Using Certitude™

- Some obvious ones
  - Identify and address VE implementation issues
  - More value from existing simulation cycles
  - Decrease risk of silicon re-spin & find bugs earlier in the flow
  - Measurable verification quality
  - Resource management
  - …
- Design and VE consistency
  - Observability issues
- Unique feedback on the verification process quality
  - Address issues while everybody is on board
  - Specification or test plan issues
  - Infrastructure issues
  - Verification logic issue
    - The design is the suspect and can't be trusted

=> Leverage improvements across projects and projects generations

Thank You!

SpringSoft
Accelerating Engineers