



**Open-Silicon**  
The Science of ASICs



**I never thought I would grow up to be this formal.**

**A reflection on Formal Verification experiences**

**Tim Stremcha – May 2010**

# Agenda topics

- Definition
- Formal Verification use retrospective
- Observed benefits
- Use models across the industry
- Deployment challenges
- Formal future
- Q & A

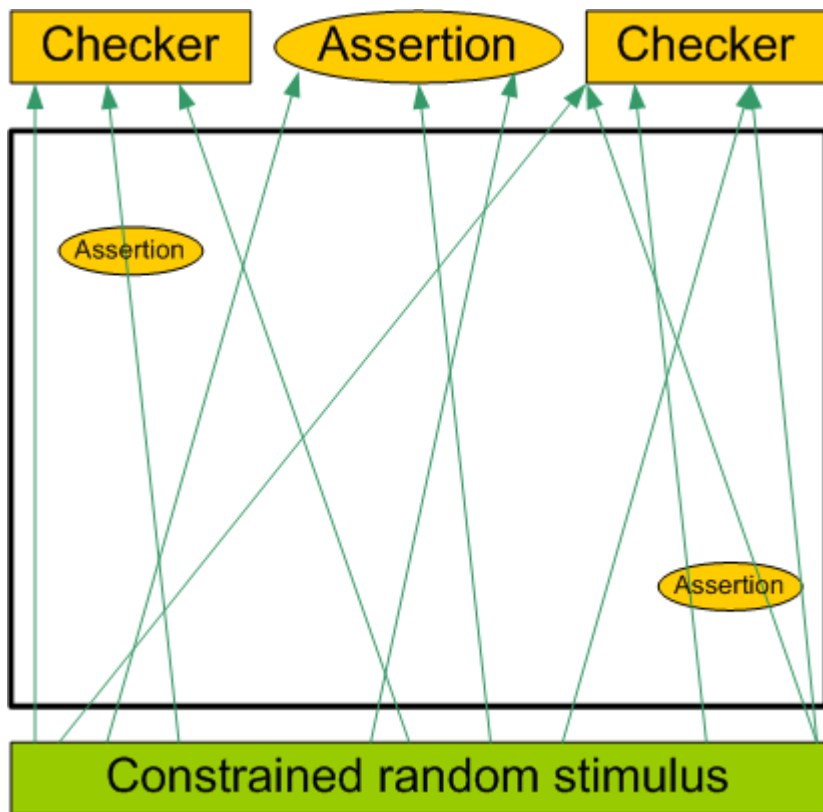
# Definition

- Formal Verification (FV) is a method of conclusively proving that a condition in a design can not be violated by legal input stimulus
  - Legal input stimulus defined with input constraint assertions
  - Checks are also inserted in the form of assertions
- Examples:
  - A FIFO will never overflow
  - bus protocol can not be violated
- This is not a discussion about formal Logical Equivalency Checking (LEC)
  - LEC has traditionally been referred to as formal verification and the name overload still causes confusion

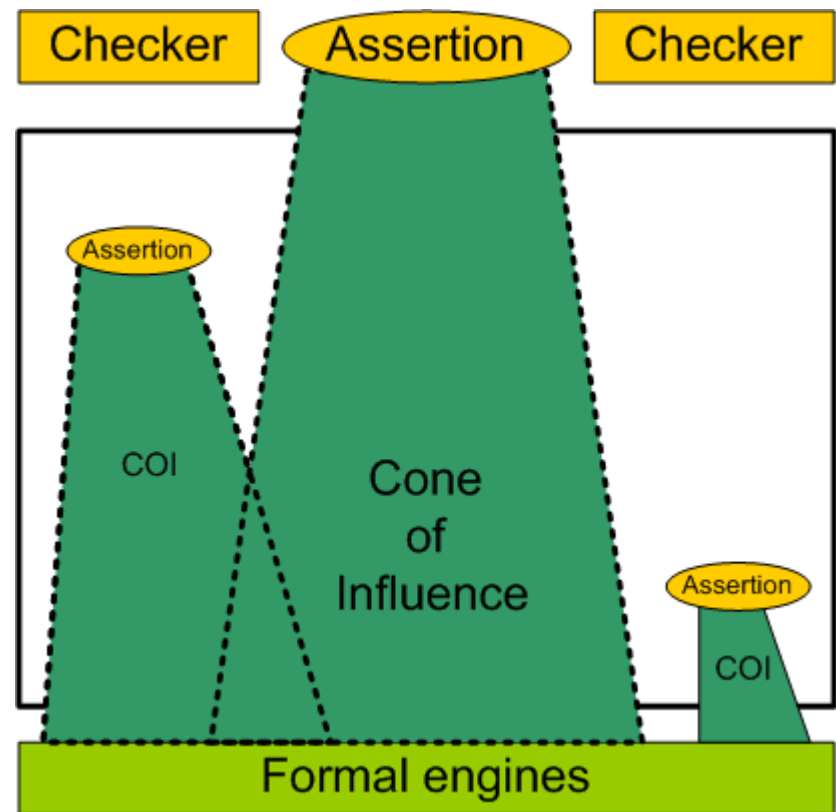
# Definition (cont)

- How is FV different from dynamic simulation based verification?
  - Dynamic simulations conclude that the range of stimulus used does not propagate a functional violation to a checker
  - Formal verification can conclude that there is no possibility of a functional violation being propagated to a checker (assertion).
    - i.e. It can prove there is no legal sequence of stimulus that can possibly violate an assertion

# Definition (cont)



- Iterative testing lowers probability of undetected flaw
- Exhaustive search is not practical in most cases



- Cones of influence to assertions are isolated and algorithmically proven to hold true or have exceptions

# Formal Verification retrospective

- Why look back?
  - It helps gain perspective on the trajectory of FV technology which can help us assess where it's most likely to be useful today, and also help predict where it's going
  - It provides a context to relay some direct experiences with some FV use models
  - It reveals the roots of some common reservations about FV and helps us weigh their relevancy today

# Formal Verification retrospective

## (cont)

### Disclaimers:

- These reflections are based on my experiences. This is not a comprehensive treatise on the evolution and use of FV nor is it a comparison of capabilities between vendors.
- Most of my work with FV has been oriented toward improving the design process efficiency and accuracy, with considerably less emphasis on final verification sign-off and coverage closure. While there is a great deal of overlap between these goals, the ability of FV to contribute more directly to the latter is underrepresented here.

# Formal Verification retrospective

## (cont)

### 1) Early FV use – circa 1996

#### ➤ Backdrop:

- Testbenches were generally a mix of straight Verilog, PLI models, and proprietary solutions
- Design synthesis from RTL was a relatively new norm
- FV tools for hardware design were in their infancy

#### ➤ Environment:

- Employed esoteric and proprietary constraint/assertion/query languages
- Could not be leveraged by the companion workhorse simulation testbenches
- Required top-notch resources which were removed from core contribution efforts
- Woefully limited capacity

# Formal Verification retrospective

## (cont)

### 1) Early FV use – circa 1996

#### ➤ Results/conclusions:

- FV gave very low return in terms of actual DV or design process efficiency improvement
- Very high monetary and time costs
- Provided negative stereotypes of FV that persist today
  - FV is “weird”
  - It requires sophisticated specialization
  - It’s questionable whether it will provide much benefit
  - It doesn’t integrate very well with the rest of the design and verification process – it’s a separate effort

# Formal Verification retrospective

## (cont)

2) Give it to the gurus – circa 2005

➤ Backdrop:

- FV was trickling into design/DV flows though fewer than 30% of companies had any FV use according to a Deepchip survey at the time
- The major CAD vendors were actively investing and staking claims in FV through acquisitions and/or organic development.
  - Cadence acquired Verplex (Blacktie) in 2003
  - Synopsys made full customer debut of Magellan in 2004
  - Mentor Graphics acquired 0-in in 2004
  - Some “indy” tools stood tall including Jasper and Real Intent
- **Recent standardization of assertion languages meant that FV preparation (constraints and checking assertions) provided value in the dynamic simulation environment**

# Formal Verification retrospective

## (cont)

### 2) Give it to the gurus – circa 2005

#### ➤ Environment:

- Specialists with highly sophisticated knowledge of FV algorithms and processes were hired to provide FV capabilities to the team by owning the tool setup and operation
- This specialized group reached in to provide verification in parallel with the DV team by targeting every assertion.

#### ➤ Results/conclusions:

- Provided real value to the design and verification processes, but relied on significant specialized expertise
- Successful at starting to bridge the gap between FV as an academic expedition and a practical application within real design flows
- Separation of skills between FV enablers and the design/DV team was too wide for widespread deployment

# Formal Verification retrospective

## (cont)

3) Designer owns it all – circa 2006

➤ Backdrop:

- Standardized assertion use by designers was becoming more commonplace
- **A large portion of the FV enabling work (i.e. the assertion implementation) was coincidentally being undertaken based on its own merits and value**
- The FV tools were maturing and reaching out to the design/dv community in a more user friendly way.

# Formal Verification retrospective

## (cont)

3) Designer owns it all – circa 2006

➤ Environment:

- Verification of the block was to be done by the customer, but wasn't available during the block design, so FV was the sole DV method used during that part of the design process
- The designer implemented a full set of assertions for the design, set up the formal environment, and ran the tool
- The design was quite configurable via control registers which presented a verification challenge in any environment. With FV all configurations were solved simultaneously.

# Formal Verification retrospective

## (cont)

3) Designer owns it all – circa 2006

➤ Results/conclusions:

- Quite effective. No logic bugs were found during post-FV verification
- Provided a tight, efficient bug identification and repair loop for the designer
- You still probably don't want to try this at home
  - Independent verification was only achieved during very late stage integration testing. This propagated too much risk forward for universal adoption.
  - Loading all FV related tasks onto logic designers unnecessarily lumps it onto a critical path resource

# Formal Verification retrospective

## (cont)

### 4) FV enabled designer – circa today

#### ➤ Backdrop:

- Robust assertion methodology in use within our design group
- More of the team is familiar with the application of FV
- FV tools continue to improve capacity and usability

#### ➤ Environment:

- FV “enabler” supports each project
  - Responsible for assertions on some or all block-to-block interfaces
  - Sets up initial FV tool environment for each designer
  - Provides first line of support to users
  - These tasks can be shared across multiple people – flexible load sharing
- Could be considered a guru-lite approach
- Each designer has near push-button access to some level of FV

# Formal Verification retrospective

## (cont)

### 4) FV enabled designer – circa today

#### ➤ Results/conclusions:

- Provides the tight bug identification and repair loop of the “Designer does it all” model, but allows better work sharing
- Specialization is limited and the work on the interface assertions are leveraged by the FV environments as well as the dynamic testbenches
- Allows a path for FV proliferation without excessive overhead on every user
- Can be supplemented by dedicated FV work for difficult properties that are beyond the capabilities of newer users

# Observed benefits of using FV

- Check of the checkers
  - The formal tools are very proficient at identifying underconstraint situations and providing short counter-example traces
  - This function is underrated and is even sometimes counted as solely a FV startup cost and therefore a FV detractor!
  - Better input constraints mean better checks on the driving block, whether that's another block or the testbench
- More granular ad hoc test environments
  - Replaces designer “preverification” test environments
  - Decouples initial testing from the availability of other design blocks and the full testbench
- Closed loop, designer controlled testing (stealthy benefit)

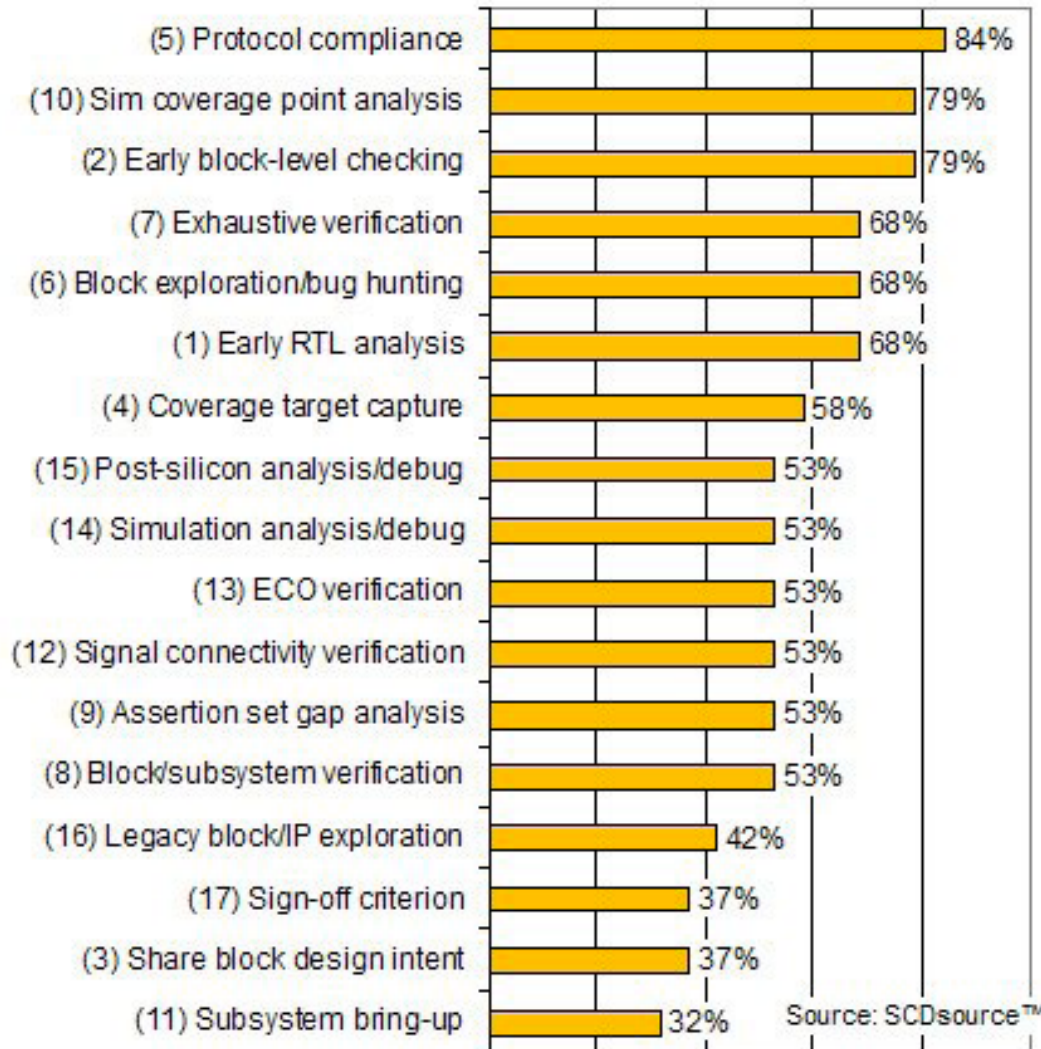
# Observed benefits of using FV

- **Higher quality input constraints, design and checker assertions enter the dynamic simulation environment**
  - Expectations and definitions are unambiguously conveyed to verification and other designers via full input constraint set
  - Illegal stimulus flagged immediately at the inputs of blocks during dynamic simulation and IP integration (stealthy benefit)
- Bounded proofs and FV bug hunting offer orthogonal testing and assurances relative to dynamic simulations.
- There's nothing like a proof
  - Exhaustive proof capability was an original core offering of FV and is still a main attraction

# Use models across the industry

Use Case Employment 2009

All Respondents (19)



From “*Mixing Formal and Dynamic Verification*” by Bill Murray

## Respondent companies:

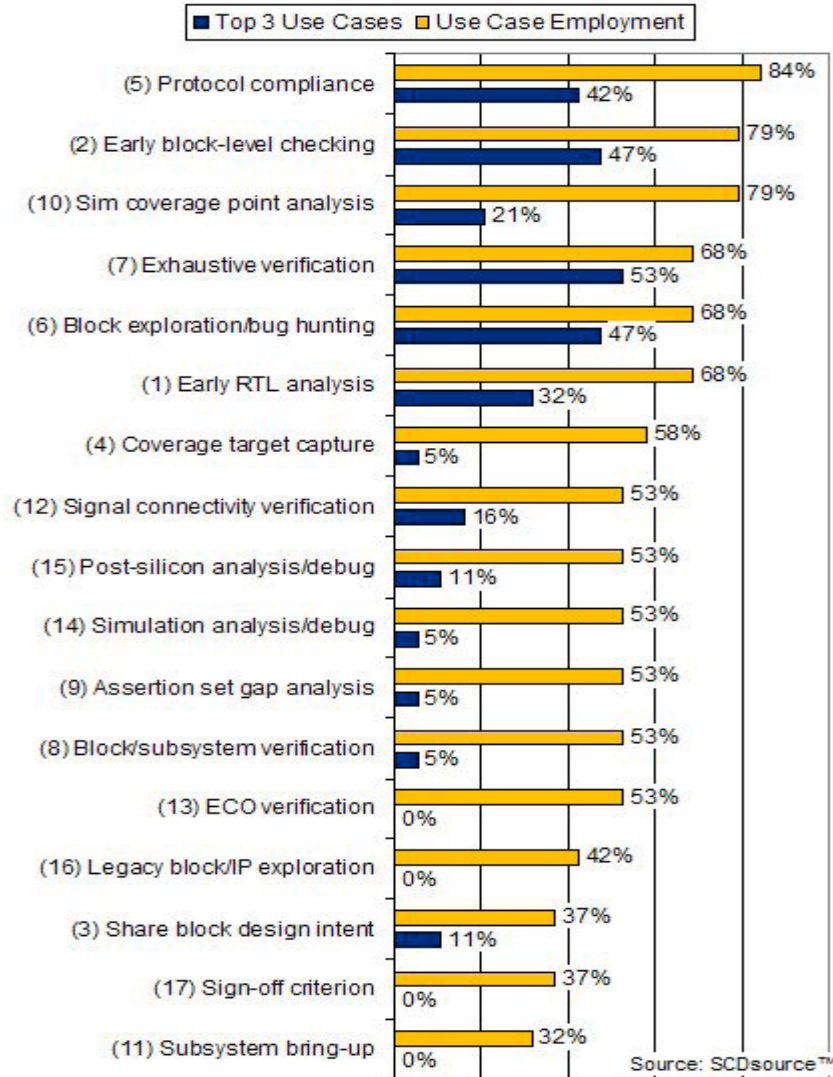
Alcatel-Lucent, Analog Devices, ARM, Cisco, DE Shaw Research (DESRES), Fujitsu Microelectronics Europe, HP, IBM, Infineon, Intel, nVidia, Qualcomm, Saab, Silicon Logic Engineering (Tundra), STMicroelectronics and Sun Microsystems

Disclosure: There is some element of self fulfilling consistency since I was one of those surveyed.

Note: numbers in parenthesis on the left indicate use model index, not respondent counts

# Use models across the industry

Use Case Employment 2009  
General Usage Compared With Top 3 Use Cases  
All Respondents (19)



From *“Mixing Formal and Dynamic Verification”* by Bill Murray

My apologies for the eye chart!

The main point is the high value attributed to use during early design (i.e. Early block-level checking and Block exploration/bug hunting) juxtaposed with no votes for late stage Sign-off criterion.

# Launch and deployment challenges

- It's a change.
- Lingering perception that FV requires super-specialized skills and is narrowly applicable.
- Requires multi-team learning, and cooperation. Fortunately, standardization has made this knowledge portable and likely to be valuable for the foreseeable future
- The discipline and completeness that FV enforces on the enabling work can be frustrating
  - It's easier to create “pretty good” constraints than to build full constraints. The extra effort to close that gap doesn't always seem worth it on the surface

# Launch and deployment challenges

- Management buy in
  - Enabling work (constraints, assertions) is sometimes perceived as simply FV overhead
  - Adoption is misperceived as “all or nothing”
    - Assertion methodology can be adopted and improved without regard to actual FV use
    - FV can be applied gradually through use on select blocks, though the tool cost is a step function
  - Some important benefits are stealthy
  - Resources are always scarce and “a bird in the hand” contributing directly to a more traditional methodology is often more attractive than an assignment to something new and less well understood
  - Money for something “we didn’t need in the past”

# Formal Future, continuing up

## ➤ Where will FV go from here?

Formal Verification will...

- Gain capacity and capabilities
  - Based on FV research advances (algorithms, programming efficiencies, etc)
  - By fully absorbing the power of the compute capacity wave
  - Faster per human resource than dynamic testbenches/tests?
- Become more seamlessly enlisted in the [OUV]MM type testbench environments
  - Some level of constraint language amalgamation would be helpful
- Continue to be more widely adopted. The more immediately observable curve will be the expanded use of the enabling assertions and constraints.
- Become essential to your methodology

Formal Verification is good.  
There's Proof!