**Formal Verification**

Formal verification checks exhaustively if a model meets a given spec.

* Model: synthesizable RTL
* Spec: properties

For more detailed mechanics of Formal: <https://ptolemy.berkeley.edu/projects/embedded/research/vis/doc/VisUser/vis_user/node4.html>

**Formal vs. Simulation**

|  |  |
| --- | --- |
| **Simulation** | **Formal** |
| User creates given stimulus set | User specifies only illegal stimulus |
| TB is a wrapper around design which drives stimulus | TB is a set of properties connected to design |
| Run tests repeatedly with the same seeds | Specify desire behaviors to observe |

* Formal has very little randomization due to its systematic method. Plus, formal finds bugs from a different angle than sim. It usually reveals bugs that simulation would hardly catch.
* Less testbench effort required: Formal testbench tends to be much simpler than sim testbench.
* Formal improves productivity and schedule. It can replace some block-level testbenches. Verification can begin prior to testbench creation and simulation.

**Formal Analysis**

Formal test all possible stimulus, one cycle at a time.

* All value combinations in inputs and undriven wires at every cycle
* All value combinations on unitialized registers at first cycle

**Properties**

Formal checks properties as it is walking through stimuli.

* Report when **assert** properties are violated
* Report when **cover** properties are hit

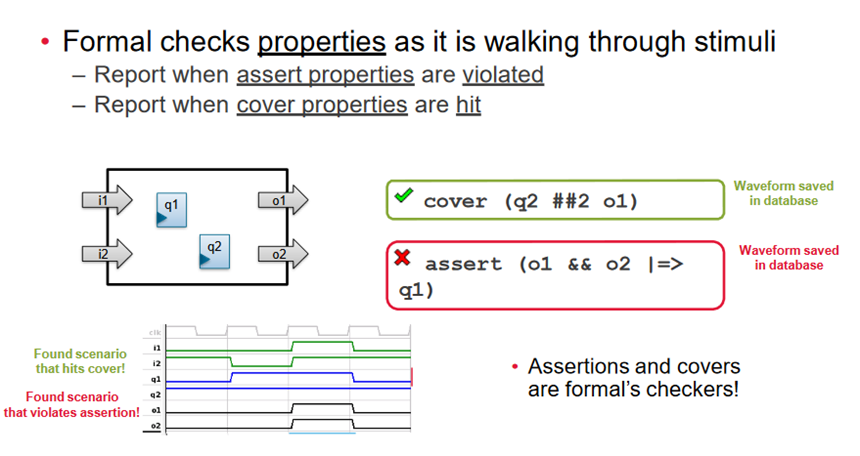


Figure 1

In Figure 1, the cover property covers the case that q2 is set AND after 2 cycle o1 is set, and the assert property asserts that if o1 and o2 are set, then on the next cycle q1 must be set. According to the waveform, the cover is hit, and the assert is violated.

\*The properties are usually written in SVA (SystemVerilog Assertion)

**Constraints**

Not all input stimulus combinations are legal. **Assume** properties tell formal what is legal.

* Only scenarios where assumptions are true will be considered
* Formal throws away any stimulus that violates assumptions

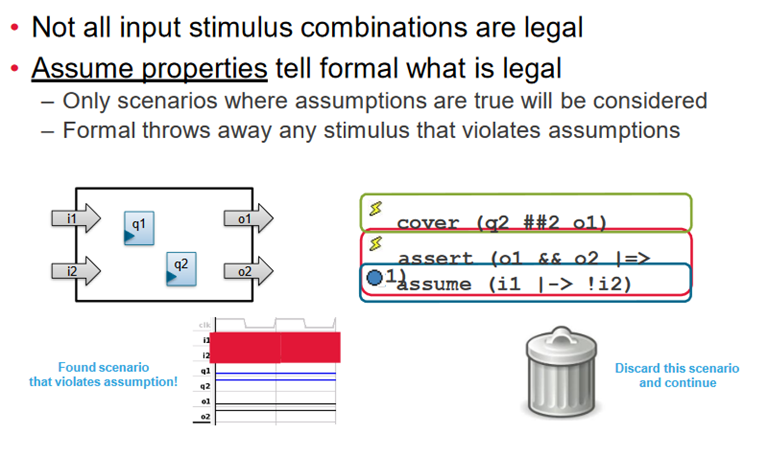


Figure 2

In Figure 2, the assume property assumes that if i1 is set, then after 1 cycle i2 must not be set. According to the waveform, the assume passes.

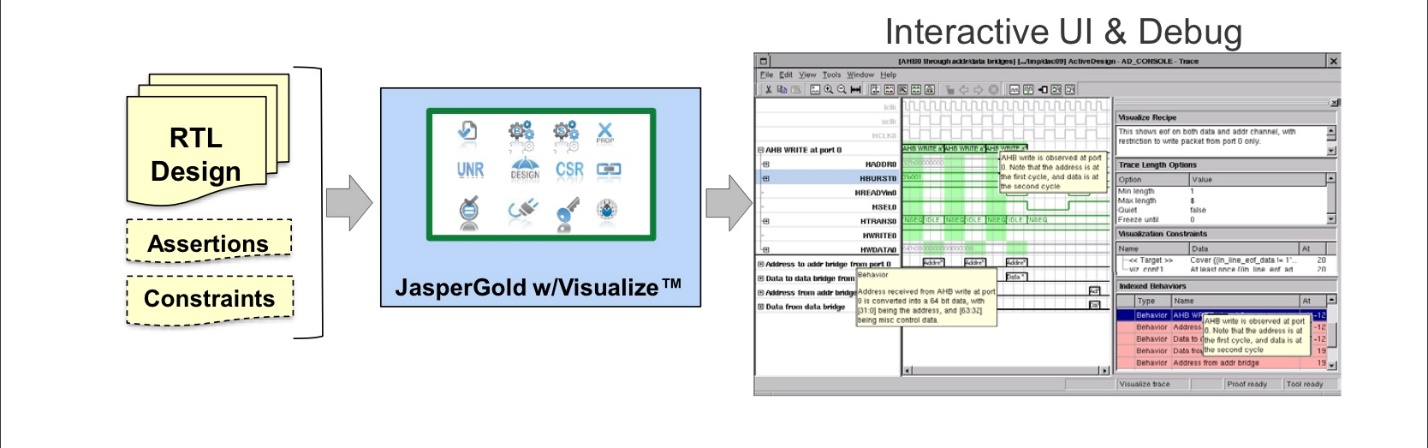
Adding constraints reduces the size of the state space, which may be crucial for verifying large design to reserve system memory.

**JasperGold**

JasperGold FPV App is the flagship app which is used for doing formal verification of the RTL. It is well suited for designs that have complex control logic, significant concurrency, complex protocols, and/or non-transforming data paths.

* JasperGold can easily be used to quickly and thoroughly explore design behaviors without any simulation testbench.
* Often, it takes a minimal time to compile a design and start exploring behaviors.
* Iterative experiments are rapid, without the bulk or complexity of a testbench.
* There is no “throwaway” work. Properties created at the block level can be handed off to the other designers or the verification team for use in both simulation and formal.
* In a team setting, properties developed at the inputs and outputs of the design block can be shared for block-level design bring-up of verification of the connecting block.

**JasperGold FPV App Flow**

****

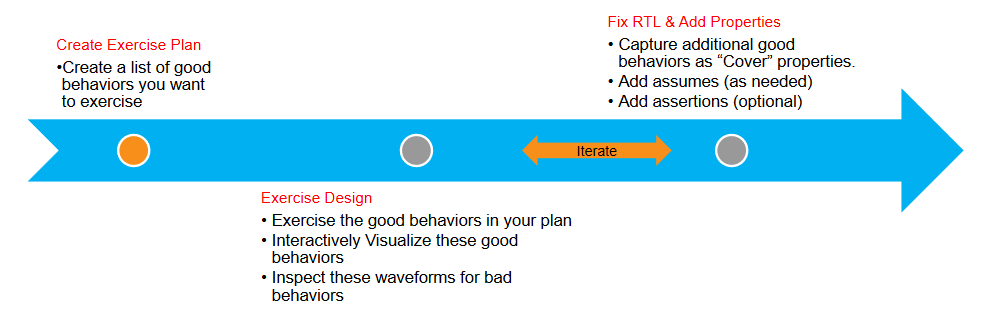
Visualize the debug environment:

* Understand waveform with advanced debugging features
* Change waveform with features that are only possible with formal

Usage examples:

* With properties: Use Visualize to debug an assertion failure
* Without properties: Use Visualize to generate waveforms which exercise interesting design behaviors.

**RTL Design Bring-Up Flow**



To be successful with RTL design bring-up in JasperGold, you need to first create a plan consisting of the good behaviors you want to exercise. You want to see that these good behaviors can be exercised, and you want to study these waveforms.

A good exercise plan should include:

* Reachability
  + All FSM states and transitions are reachable
  + All valid output states are reachable
  + Configuration bits can be programmed
* Functional Sequences
  + Archetypical sequences to test the integrity of the design

As you exercise the design, you may capture additional behaviors as properties that can be handed off to other simulation and formal teams.

You will iterate between exercising and capturing properties until you are able to observe all good behaviors in the plan.

**Design Bring-Up Showcase**

Video Link:

I apologize for the occasional discontinuities of the video due to editing to cut out background noise.

**Source**

Kenji, KJellson. *Formal Methods in Verification of Interface and Bus Protocols*. KTH Royal Institute of Technology, 2014. (<http://www.diva-portal.se/smash/get/diva2:872375/FULLTEXT01.pdf>)

Cadence Support: JasperGold Formal Property Verification App (FPV).

Cadence Support: JasperGold RTL Design Bring-Up Rapid Adoption Kit

**Credit**

Yiming (Billy) Li

John Martinuk

Mark Johnson