Leveraging Formal Verification Throughout the Entire Design Cycle

Verification Futures

Objectives for This Presentation

- Highlight several areas where formal verification has been successfully used throughout the design cycle
- Provide some insight for identifying good opportunities for applying formal verification for maximal ROI
- Show some of the innovations in formal verification that have enabled broader adoption and higher project benefits

About Jasper

- Jasper Design Automation
  - Leading provider of SoC design and verification solutions leveraging advanced formal technologies
- Jasper Users
  - Include system architects, logic designers, verification engineers, and silicon bring-up teams
- Jasper's Success
  - Our year-to-year growth based on successful, proven technologies; excellent AE support; and deployment-driven business model

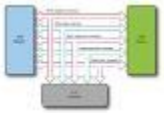
What Is the Perception of Formal Verification?

- It is a point tool
- One needs to have a PhD to use it
- Verifies only module/block-level RTL
  - Can verify only small portions of the design (e.g., FIFO overflow)
- Need to write 100s/1000s of properties
  - Need to learn a new language to do this
- Involves a deep learning curve on property languages
- Debugging failure traces is difficult and time consuming

This perception is not the reality!

What Is Really Possible with Formal

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes



Intelligent Proof Kits and Verification IPs
- Certification of AMBA 4/ACE checkers
- Popular standard protocols
- Configurable, illustrative, optimized for formal

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

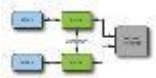
Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

X-Propagation Verification
- Unexpected X Detection and debugging

CSR Verification
- Automated register verification



Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

… and many more

What Is Really Possible with Formal

Formal Property Verification
- •Traditional application of formal
- •More than just block-level checks

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simula
- Identification of coverage holes

- Configurable, illustrative, optimized for formal

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

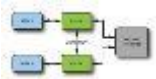
Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

X-Propagation Verification
- Unexpected X Detection and debugging

CSR Verification
- Automated register verification



Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

… and many more

What Is Really Possible with Formal

Formal handles both x-optimism and x-pessimism, when simulation is not helping

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

- Configurable, illustrative, optimized for formal

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

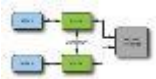
Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

X-Propagation Verification
- Unexpected X Detection and debugging

CSR Verification
- Automated register verification



Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

… and many more

What Is Really Possible with Formal

Formal increases SoC integration productivity

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

- Certification of AMBA 4/ACE checkers
- Popular standard protocols
- Configurable, illustrative, optimized for formal

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

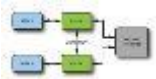
Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

X-Propagation Verification
- Unexpected X Detection and debugging

CSR Verification
- Automated register verification



Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

… and many more

JASPER design automation

What Is Really Possible with Formal

Formal provides visibility into a design, isolating relevant areas effectively

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

- Popular standard protocols
- Configurable, illustrative, optimized for formal

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

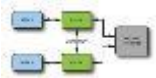
Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

X-Propagation Verification
- Unexpected X Detection and debugging

CSR Verification
- Automated register verification



Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause isolation
- Candidate cause elimination
- Validation of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

… and many more

What Is Really Possible with Formal

Property Synthesis (Structural / Behavioral)
- Automated assertion generation
- Functional pre-defined property generation
- Inference & synthesis of properties from RTL & simulation
- Identification of coverage holes

Intelligent Proof Kits and Verification IPs
- Certification of AMBA 4/ACE checkers
- Popular standard protocols
- Configurable, illustrative, optimized for formal

Formal Property Verification
- Protocol certification
- End-to-end packet integrity
- Asynchronous clocking effects
- Assertion-based verification
- Proofs for critical functionalities
- Debug isolation and fix validation

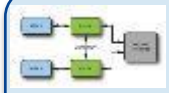
Executable Spec
- Design IP documentation
- Cross references among document, waveform, and RTL
- Configurable waveforms

RTL Development
- Waveform generation from intent
- Designer-based verification w/o testbench
- Design trade-off analysis

Connectivity Verification
- Chip-level connectivity
- Conditional connection with latency

X-Propagation Verification
- Unexpected X Detection and debugging

CSR Verification
- Automated register verification



Architectural Modeling
- Executable spec
- Absence of deadlock
- Cache coherency

Post-Silicon Debugging
- Failure signature matching
- Root cause is
- Candidate ca
- Validation of

Other SoC-Related Applications

Synergy from various sources of properties at various abstraction levels

… and many more

What Is Really Possible with Formal

Architecture Modeling

RTL Development

Functional Verification

SoC Integration

Post-Silicon Debugging

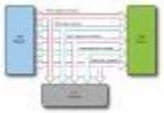
Formal Verification throughout the Entire Design Cycle

Prop
- Au
- Fu
- In
- Id

Form
- Pr
- Er
- As
- As
- Pr
- De

Connectivit
- Chip-leve
- Conditio

nulation



Intelligent Proof Kits and Verification IPs
- Certification of AMBA 4/ACE checkers
- Popular standard protocols
- Configurable, illustrative, optimized for formal

ntent
v/o

- Automated register verification

xpected x Detection
debugging

on Debugging
signature matching
ause isolation
late cause elimination
tion of fixes before re-spin

Other SoC-Related Applications
- Glitch detection
- Multi-cycle path verification
- Low power verification

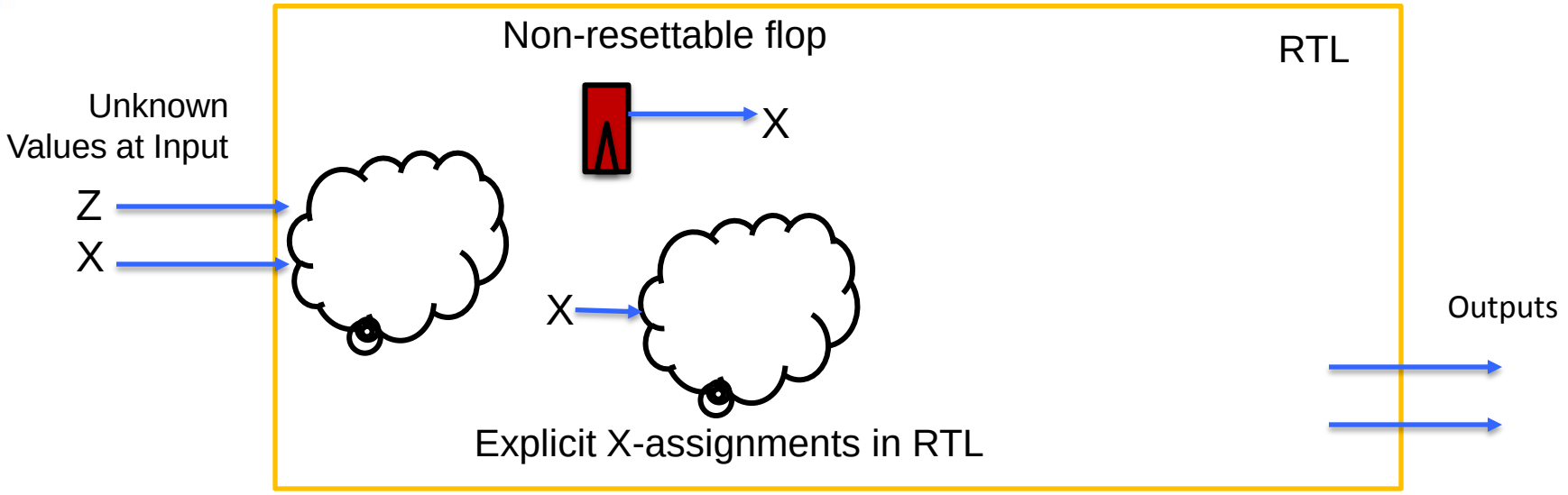
JASPER design automation

Formal Verification throughout the Entire Design Cycle

X-PROPAGATION VERIFICATION

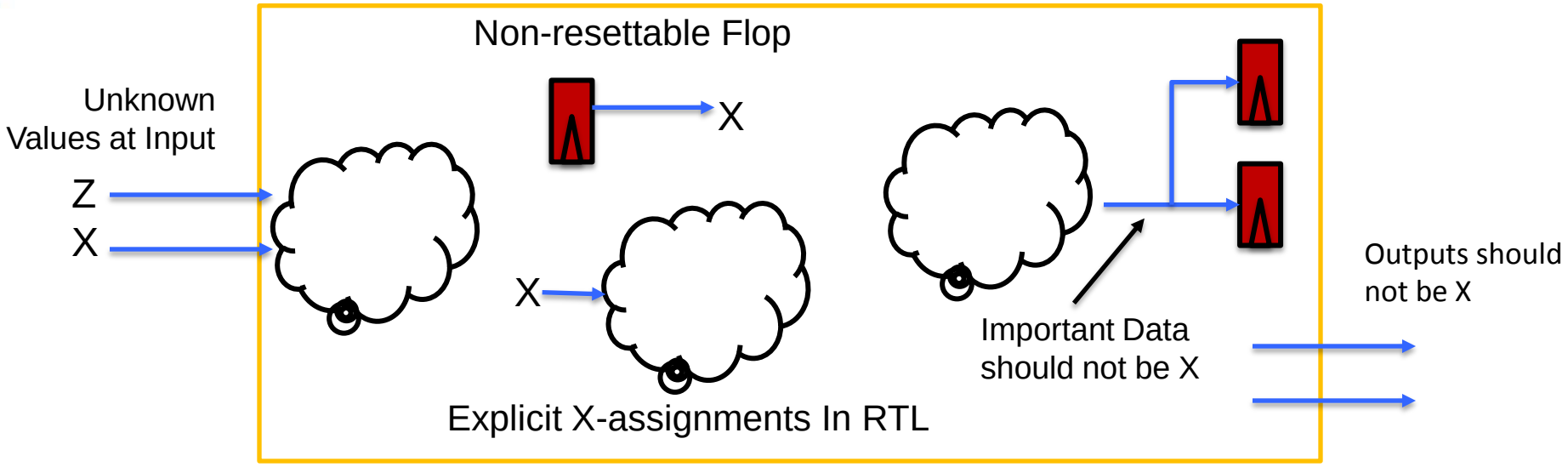
JASPER design automation

Where Do Xs Come From?



- Unknown values at design inputs
  - Check input values and propagate Xs if needed
- Non-resettable memory elements
  - Expensive to make all elements explicitly resettable
  - RTL intent is that “write” occurs before any “read”
- Explicit X-assignments in RTL
  - For optimization purposes (e.g., some address bits are “don’t care” under some conditions)
  - To properly propagate Xs to upstream logic to catch Xs with proper checker in simulation

Detecting Unexpected X-Propagation



- Cannot rely on simulation to detect unexpected X propagation
  - Simulation behavior of X does not accurately portray the behavior of the circuit
  - Simulation is not exhaustive
- Formal can be used, if configured properly
  - `$isunknown` construct in SystemVerilog Assertion language (SVA)
  - Special formal engines with correct X semantics, not just Boolean formal engines

X-Propagation Validation with Formal

- Exhaustively checks whether Xs can propagate to some target signals
  - Formally optimized treatment of “X” with “smart-x-modeling”
    - Avoids performance overhead of brute-force, 3-valued analysis
  - Xs are treated as either 0 or 1, reflecting actual silicon behavior
    - No missed bugs due to either X-optimism or X-pessimism
- Functional errors detected include:
  - Unknown values propagating to output data buses for “valid” data as indicated by the data enable signals
  - Incorrect clock-gating not easily found in simulation
  - Uninitialized registers affecting control logic

Formal Verification throughout the Entire Design Cycle

SOC INTEGRATION

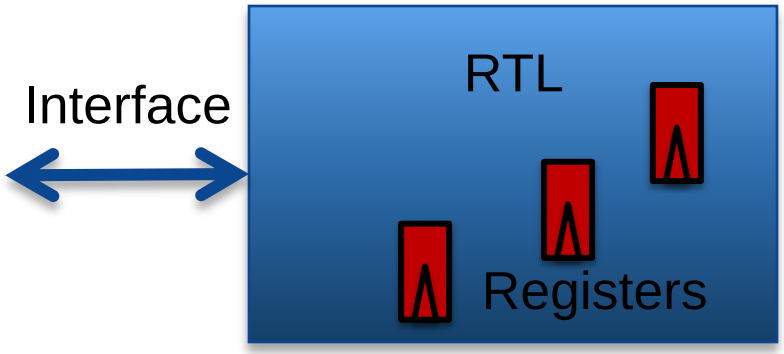
SoC Integration Verification with Formal

- Automated register verification
  - Prove data integrity of register fields and reset values
- Glitch verification
  - Identify and verify possible clock glitches in the design
- Multi-cycle path verification
  - Accurately verify multi-cycle path waivers
- Chip-level connectivity
  - Exhaustively verify that RTL matches connectivity definition
- Other applications

Register Verification with End-to-End Properties

- Given a DUV with register space accessible by:
  - Standard interface (AHB, OCP, etc.) or proprietary interface (parallel, serial)
- Automated flow provides better verification
  - Saves project time and human time
- To prove end-to-end properties such as:
  - Data integrity of register fields (exhaustive)
    - I.e., data read from a register equals previously updated data (write, reset, etc.)

Checks/assertions on programming sequence behaviors



Register Definition

May be captured in different formats:

- Spreadsheet/CSV
- IP-XACT
- Custom text format
- Etc.

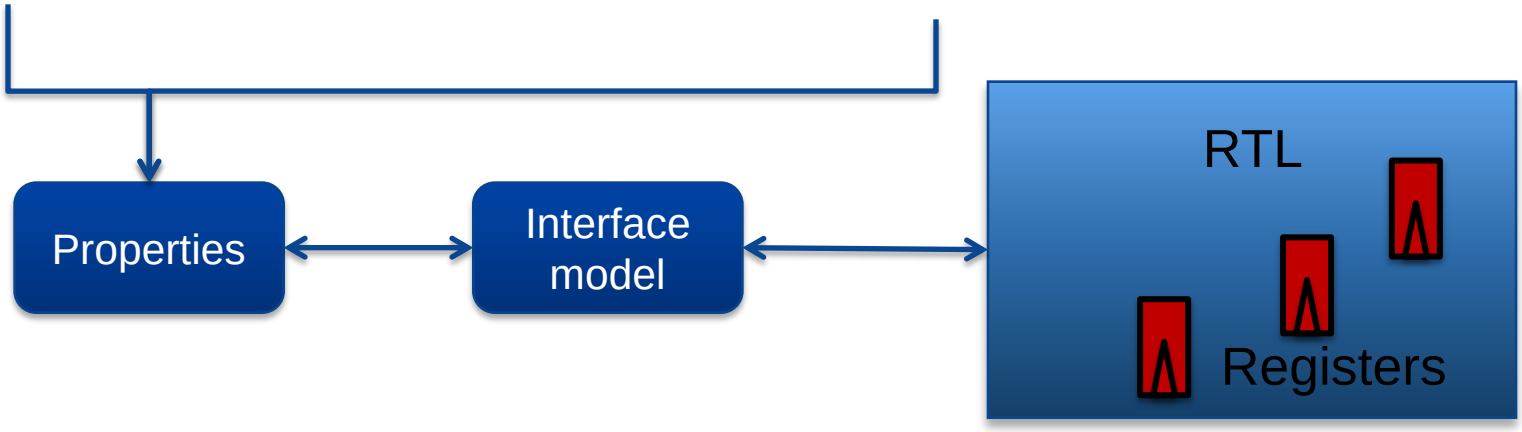
Let a tool or a script translate this into formal-friendly properties

```
            ADDRESS             0x0001C
            ACCESS_TYPE         RW
            RESET_VALUE         0x00000000
--field
      RESERVED31                BIT[31:21]
      CONS_ID                   BIT[20:16]
      RESERVED15                BIT[15:5]
      PROD_ID                   BIT[4:0]
...
```


Comprehensive Ranges of Register Behaviors

- A single register (a single address) might have numerous fields, and they can have different attributes:
  - Access types
  - Widths
  - Reset values

- Access Types
  - R: readonly
  - RW: read write
  - RS: read and set to 1
  - RC: read and clear to 0
  - RR: read and reset to reset value
  - RO: read always see value ones
  - RZ: read always see value zeros
  - Etc.

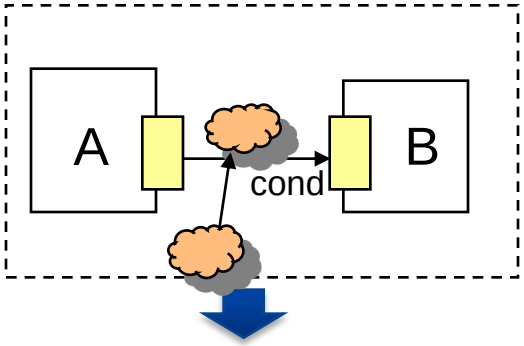


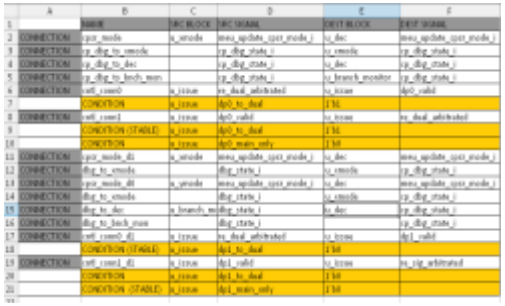
Chip-Level Connectivity Verification Solution

- Exhaustively verifies that the RTL matches the connectivity definition
  - Verify that point A is equivalent to point B (block or chip level) as certain signals/modes can impact connections
  - No other signals/modes/settings can impact connections
  - Important aspect of system integration of many IP's
- Types of connection
  - Structural, Boolean condition, temporal condition, and temporal connection with latency and delay
- Allow fast and exhaustive verification
  - Quickly reconfirm results (regressions) as RTL is being modified
  - Automated flow allows early and frequent verification

Chip-Level Connectivity Verification Flow

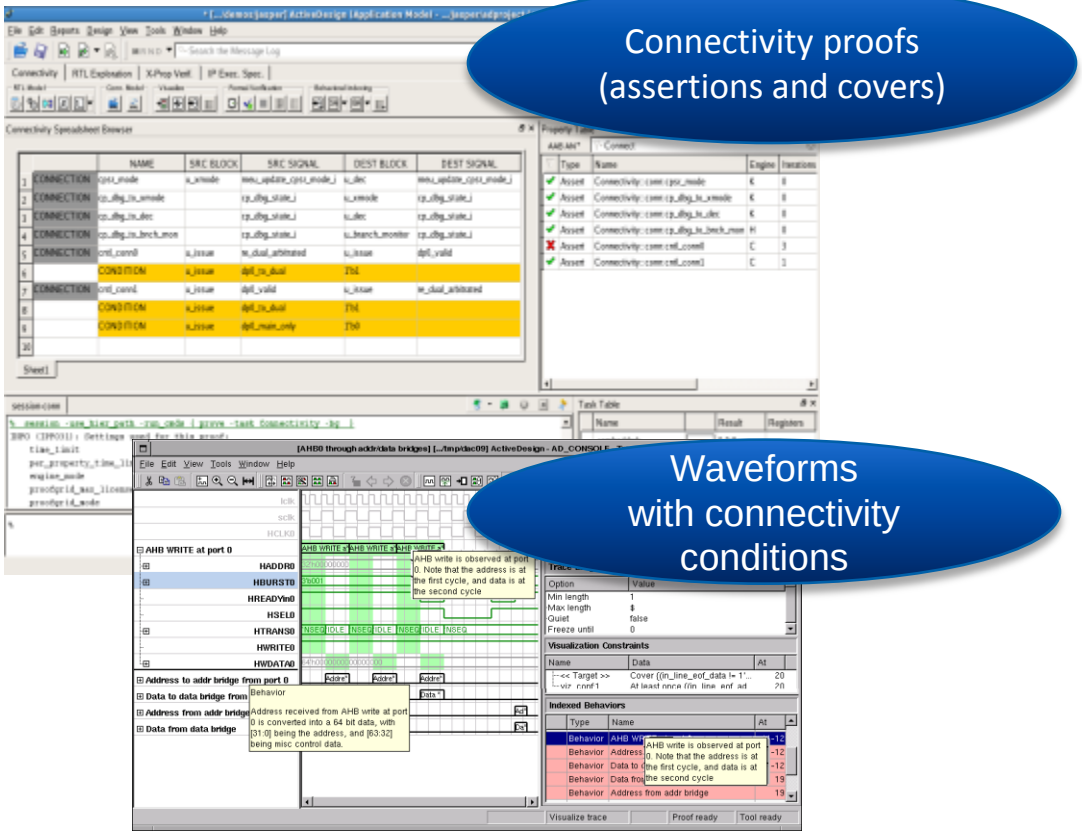
Top-level of SoC





Connectivity map

RTL



Connectivity proofs
(assertions and covers)

Waveforms
with connectivity
conditions

JASPER design automation

SoC Integration Summary

- Identify areas where automation is desired
  - Both verification time and verification resource savings
  - Exhaustive
- Areas that have been automated
  - CSR verification
  - Accurately verify multi-cycle path waivers
  - Detect glitches in the design and generate optimal set of assertions that can be used in simulation
  - Exhaustively prove that RTL matches with connectivity definition

Formal Verification throughout the Entire Design Cycle

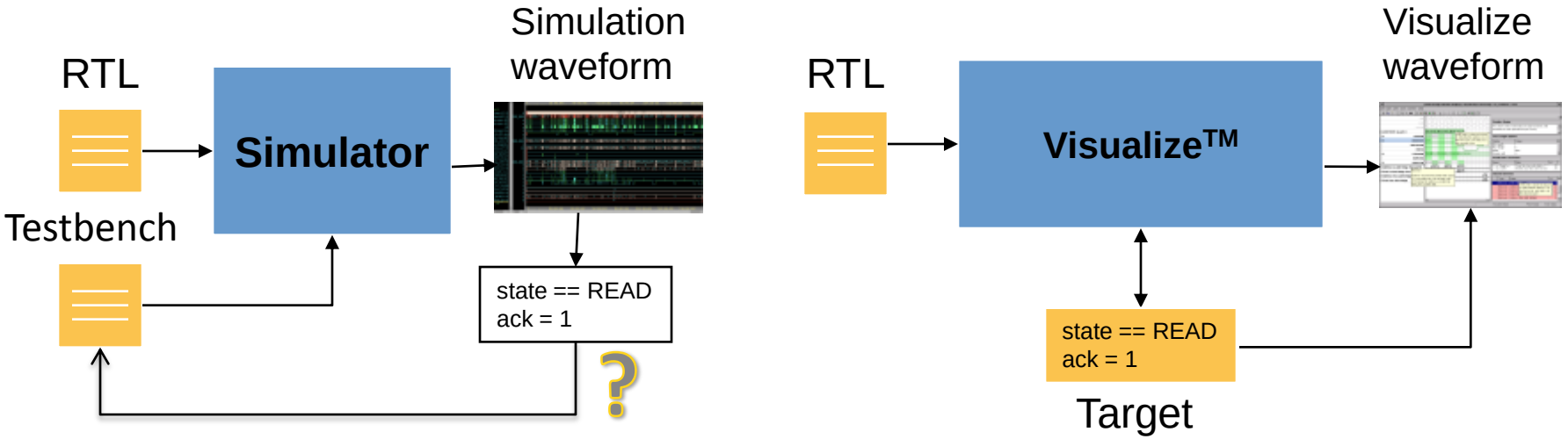
RTL DEVELOPMENT AND EXECUTABLE SPECIFICATION

Typical Designer-Based Verification

- Testbench and input stimulus are required to explore and verify design behavior
  - Usually unavailable at early design stage or smaller block levels
  - Designer does not have time to create extensive tests
- No systematic method for confirming RTL functional scenarios as each feature is added to the RTL code
  - Usually done by eye-balling the RTL
- Inability to confidently customize an existing RTL block for multiple projects

This usually means designer-based verification is not done

Rethinking Designer Verification

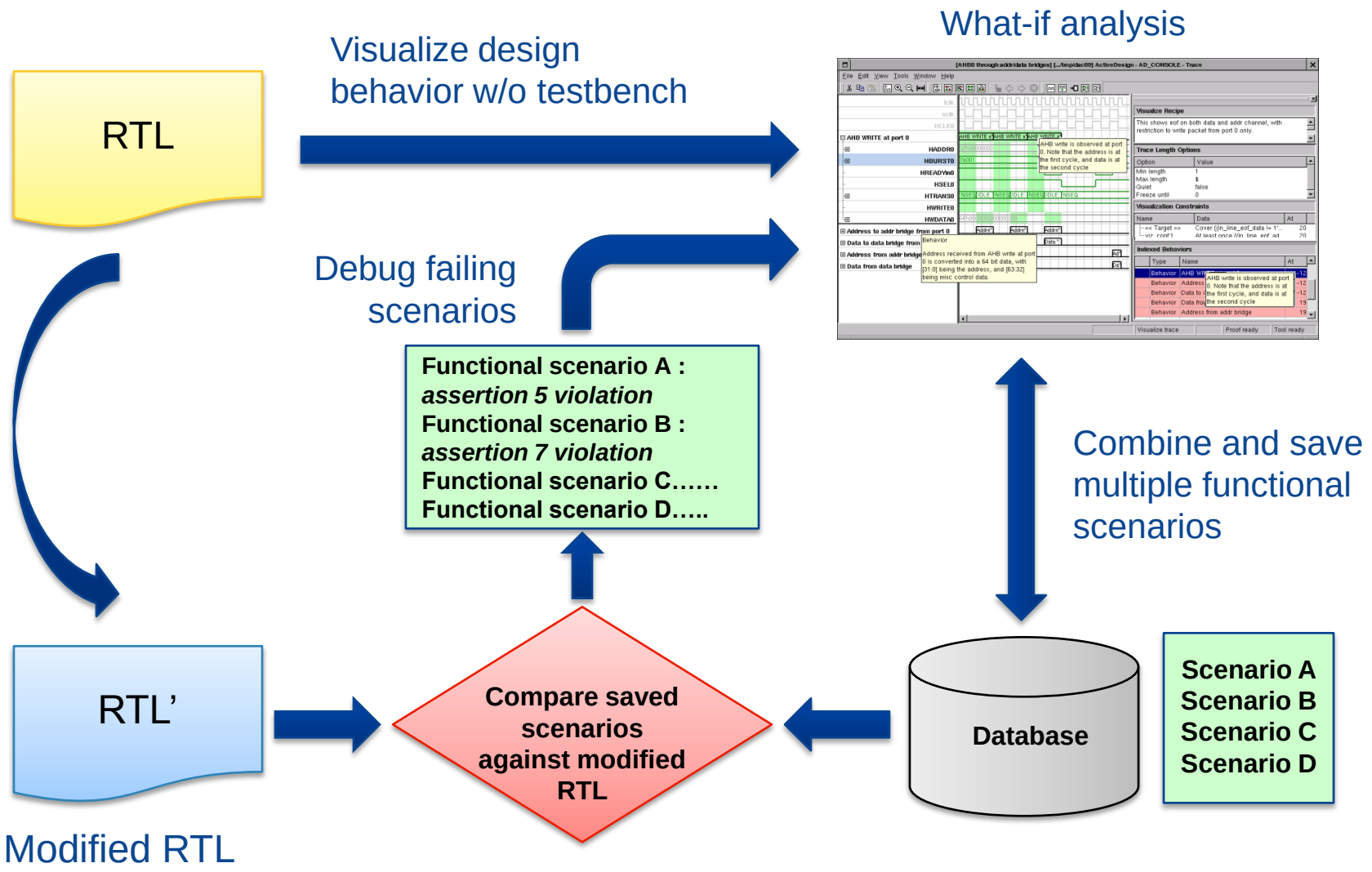


- Simulation
  - More of an “input driven” method, may not exercise desired behavior
  - Wiggle the inputs to produce a desired behavior (trial and error)
- Visualize
  - Specify the target and let the formal engines generate the stimulus (“output driven” method)
  - Interactively add constraints to construct desired waveform

Formal for RTL Development

- Designer-based verification w/o testbench
  - Allows early RTL exploration without the need to generate input stimulus
  - Start with simple behaviors about the design
    - *cover line_eop*
  - Group simple behaviors together to build complex scenarios
  - Write assertions about events that are always/never true
- Design trade-off analysis
  - Behaviors and scenarios allow for easy incremental analysis and RTL comparison tasks
- Higher quality RTL passed to other teams in the design/verification flow

Complete Flow for RTL Designers



RTL Development Summary

- Conduct early RTL exploration w/o a testbench
- Store expected functional scenarios and validate against modified RTL
- Perform design trade-off analysis while RTL is being developed
- Properties developed at this stage live with the RTL and are leveraged throughout the verification flow

Formal Verification throughout the Entire Design Cycle

PROPERTY SYNTHESIS

Properties for Design and Verification

- Critical to improve verification coverage, expose functional coverage holes
- Assertions “firing” point to bugs, reduce debugging time
  - Traditional checkers can miss bugs
  - Saves 50% debugging time, closer to RTL than checkers
- Writing properties can be difficult: it’s an “art”
  - White box: RTL designer writes
    - RTL implementation specific
    - Can overlap black box
  - Black box: Verification engineer writes
    - Integration issues for modules. Closer to Spec
- Engineer can typically only write 5-10 properties a day
  - Written correctly? – only know if used in simulation/formal

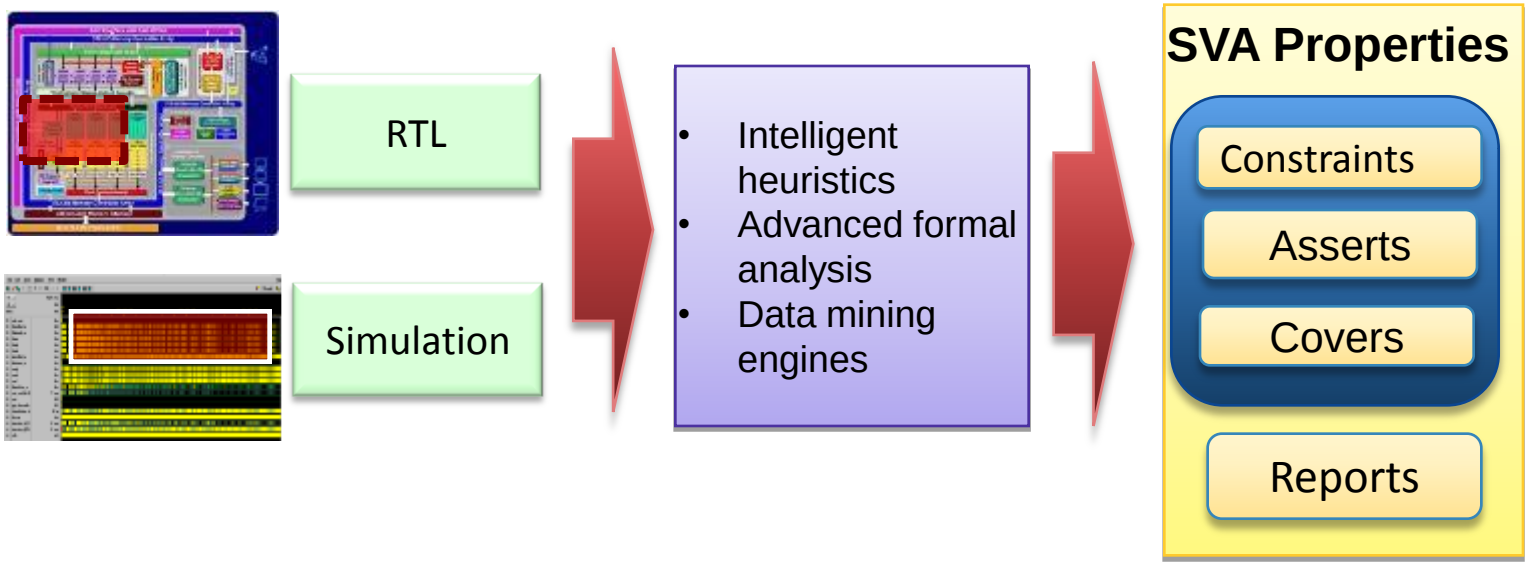
Property Synthesis

- Sources of properties
  - Structural
    - Extracted from RTL
    - No testbench required
    - Valuable during RTL development
  - Behavioral
    - Extracted from simulation (with/without knowledge of RTL)
    - Quality of properties directly tied to maturity and quality of the simulation results
    - Usually used in later stages of verification

Structural Property Synthesis

- Properties can be automatically extracted from the RTL for common structures without simulation results:
  - Non-synthesizable constructs
  - Unintentional latches
  - Out-of-range indexing
  - Arithmetic overflow
  - Full and parallel case issues (for SystemVerilog and Verilog)
  - Dead code or unreachable blocks; Stuck at signals
  - Finite state machines (FSM)
    - Livelock/deadlock states
    - Reachable FSM states/transitions
  - …

Behavioral Property Synthesis Flow



Obtain simulation results with:

- VCD/FSDB files
- Link PLI with simulator

Output SVA properties for:

- Simulation / emulation
- Formal

Behavioral Property Synthesis for Formal

- Module-interface properties:
  - Extract assumptions about the interface
  - Faster ramp-up time for the formal environment
- Multi-cycle properties (not limited to 1 or 2 cycles):
  - High value assertions that may never fire in simulation
  - Failing traces are significantly shorter and easier to debug with formal
- Cross-hierarchical
  - High-value assertions
  - Formal can prove or disprove inter-block relationships

Property Synthesis Summary

- Properties can be used as assumptions to quickly ramp up the formal environment
- Covers provide confidence in design operation and can detect overconstraints
- Formal can be leveraged during RTL design
  - Prove properties before code check-in
  - Remove common design errors before the start of validation cycle
- Should formally verify properties before including them in simulation
  - If a cover cannot be exercised with formal, then it will never be hit in simulation
  - Failure traces for assertions are much shorter and easier to debug compared to simulation

Conclusion

- Formal has been expanded tremendously over the years
  - Understanding the challenges in verification leads to great methodology innovation in formal applications
  - Integration of formal into mainstream verification flow causes many innovations in the technology to enable wide use
- By focusing on the problems and challenges, formal can be applied as part of the overall verification strategy
  - Identify areas where stimulus and coverage is the main bottleneck
  - Identify opportunity for automation to reduce project time and effort
  - Focus on high-risk areas (critical and/or new functionalities) to maximize ROI (return on investment)
  - Working closely with formal vendors to solve new problems

www.jasper-da.com