Industrial Application of Formal Verification

Magnus Björk

Jasper Design Automation

JASPER design automation

Copyright Notice and Proprietary Information

100 View St., Suite 101
Mountain View, CA 94041
Tel: (650) 966-0200
Fax: (650) 625-9840
http://www.jasper-da.com

About Me

- M.Sc from Göteborgs Universitet, PhD from Chalmers
- Continued to do research (Chalmers, Oxford, IT University, SAAB Space/RUAG)
- Now employed by Jasper for a collaboration project with Chalmers, funded by Vetenskapsrådet
- Main research interests:
  - Formal verification (algorithms and applications)
  - Automated theorem proving
- My role at Jasper
  - Research and development of new verification methods

INTRODUCTION TO JASPER

JASPER design automation

Jasper Design Automation

- Head office in California
- Other offices
  - Sweden (Gothenburg)
  - Brazil (Belo Horizonte)
  - Israel (Haifa)
  - Japan (through distributor)
  - Opening: India
- Total number of employees: ~70
  - Currently distributed ~50/50 between sales and engineering
- Gothenburg team
  - 6 employees (R&D)
  - 5 from Chalmers (many historically)

Jasper's Top 10 Customers

Sony Corporation

JASPER design automation

Jasper Products

Solution	Definition	Value
JasperGold®	Advanced formal property verification	Solves top project challenges across a spectrum of SoC applications
ActiveDesign™	Databases and analysis system for design and reuse	Accelerates design development and leverages designs and IP
JasperCore™	Formal verification solution for intelligent resource management	Deploys economically-scalable formal technology across computers and teams

JasperGold®

JasperGold® Verification System
Formal verification of design behavior
for complete correctness, clarity and closure

Visibility
Patented Visualize™: the on-ramp to formal

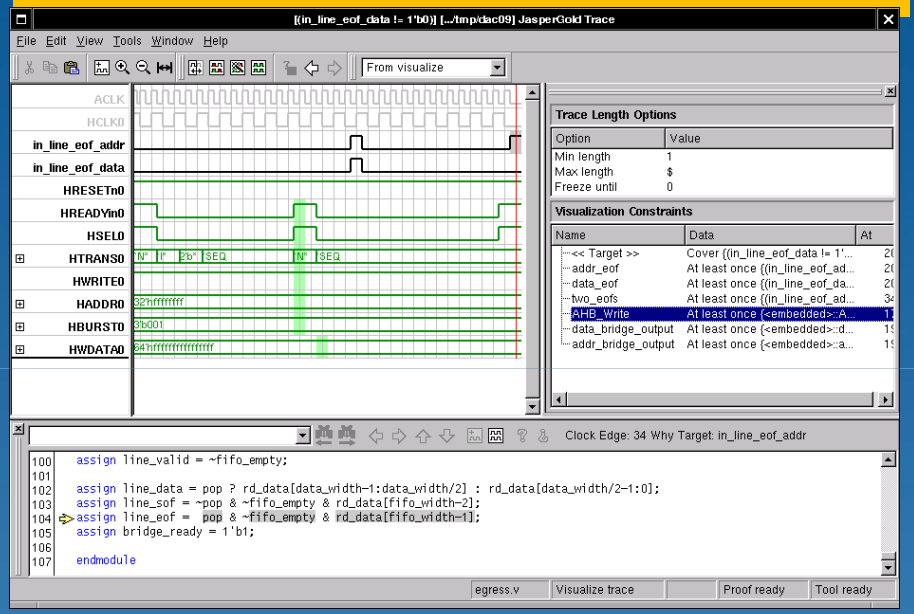
Advanced Proof Power
Fast proofs, high capacity, low memory footprint
Abstractions for greater proof capacity

ProofGrid™
Distributed, scalable formal technology for accelerating multiple proofs, tasks, users, applications, computers and productivity, even across multiple business units

Visualize™ Designs

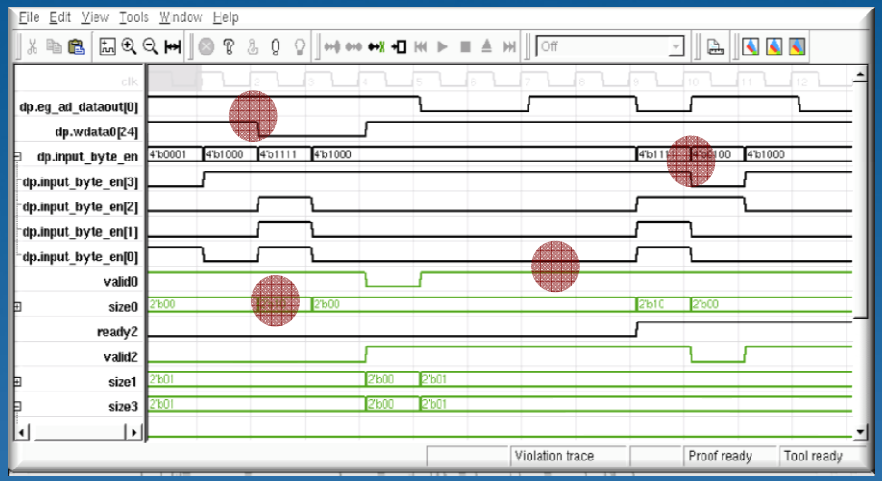
- Accelerated RTL development and debug; leveraged legacy design and IP
  - Graphical and waveform views of design functionality and dependencies
  - Shorter iteration cycles for exploration, understanding behaviors and root cause analysis
- Visualize is in both ActiveDesign and JasperGold

Visualize™

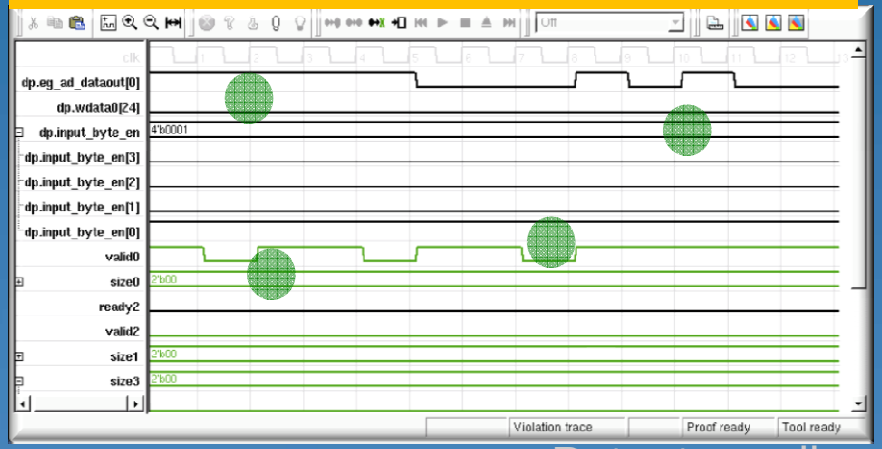


QuietTrace™

Normal Trace



QuietTrace



Patent pending

ROI: Reduced designer iterations and simplified debug

- Find similar behaviors with fewer signal trace events
  - Smooth selected traces and signals, on-demand
  - Temporal smoothing
- Visualize with QuietTrace is in both ActiveDesign and JasperGold

JasperCore™

Jasper Core™

Economically-scalable formal verification for easy deployment of powerful formal technology, across the spectrum of applications

Advanced Proof Power

Fast proofs, high capacity, low memory footprint
Abstractions for greater proof capacity

Proof Grid™

Distributed formal technology for accelerating multiple proofs, tasks, users, applications, computers and productivity, even across multiple business units

Proof Power!

Faster proofs, reduced memory footprint, higher proof capacity

- High-performance proof engines, patented tech
- Architecture for lean, fast flows
- Tunneling
- Design traversal algorithms
- Powerful abstractions: Memory, Multiplier, and Asynchronous Proof Accelerators and Scoreboard

Proof power:
in JasperCore and JasperGold

Proof Power and Capacity

ProofGrid™

Formal technology deployment for productivity

- Distributed, economically-scalable formal technology to accelerate proofs and facilitate all applications
- Parallel processing for local machines, clusters and farms
- Dynamic scheduling and engine allocation
- Seamless tracking and reporting

ProofGrid:
in JasperGold and JasperCore

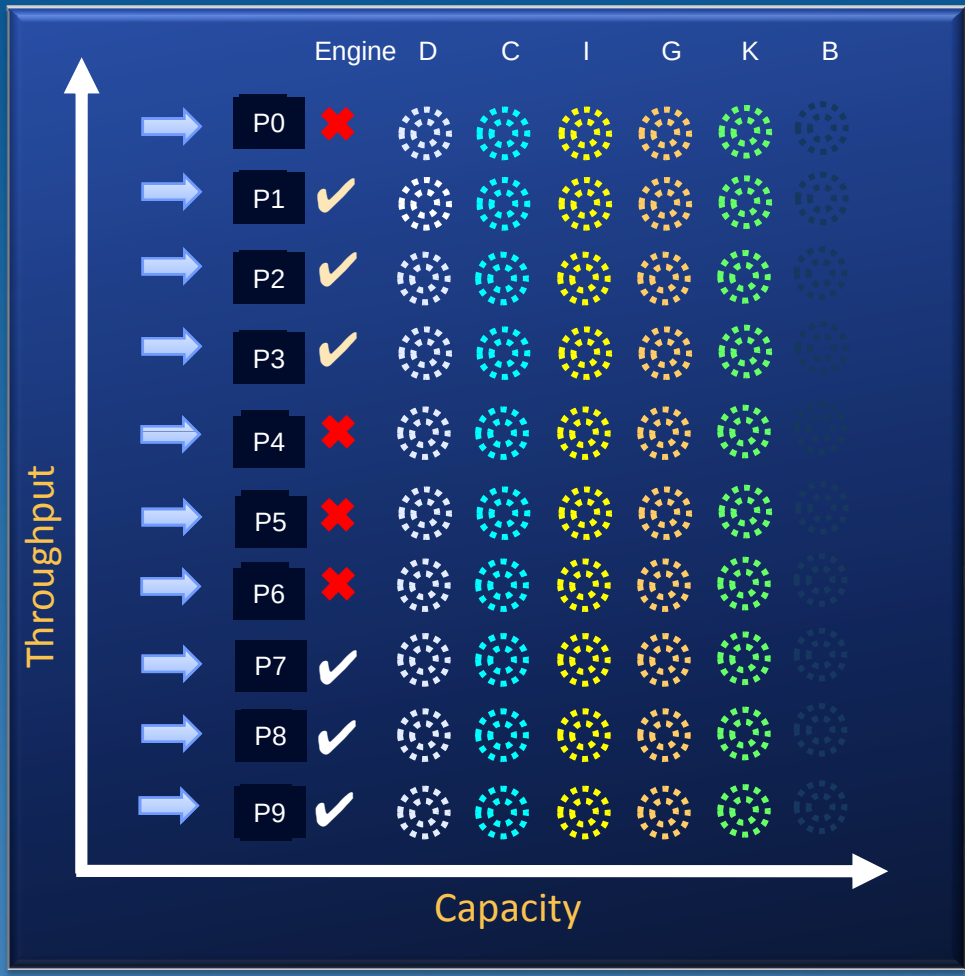
Scalable Verification

ProofGrid™

Intelligent resource management for multiple:

- Users
- Proofs
- Tasks
- Applications
- Computers
- Groups

ProofGrid:
in JasperCore and JasperGold



ActiveProp

- New product
- Reads simulation trace, proposes properties fulfilled by the trace
- Useful for developing properties
  - Easier to evaluate by example than writing your own
- Useful for developing constraints
  - When verifying how circuit interacts with external circuitry

X-propagation

- New feature in 7.2 (partially available in 7.1)
- See if X can propagate from point A to point B
- Useful for
  - Checking influence
    - Does point A influence point B?
    - Does the flop A need to have reset value?
  - Understanding
    - ”Show me a trace where the value of B depends on A”

FORMAL VERIFICATION

JASPER design automation

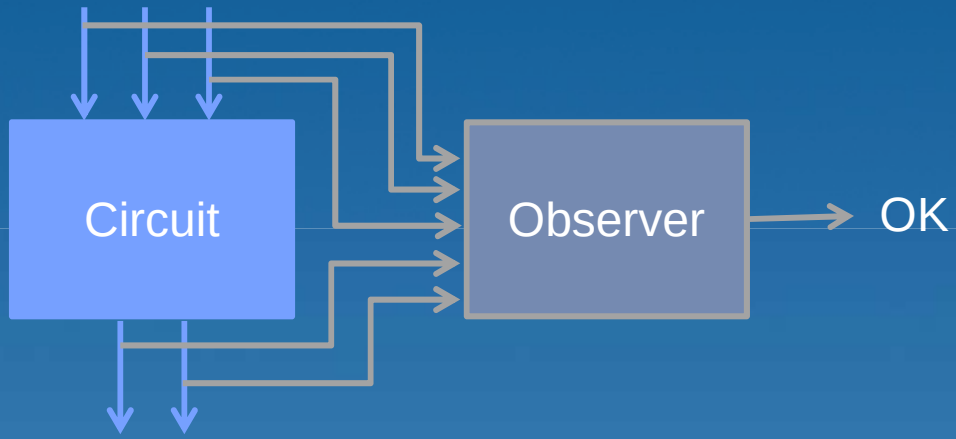
Formal verification in a nutshell

- Prove that a circuit fulfils its specification

- Otherwise: produce **counter example**
  - Trace of circuit where property is false

Using observers

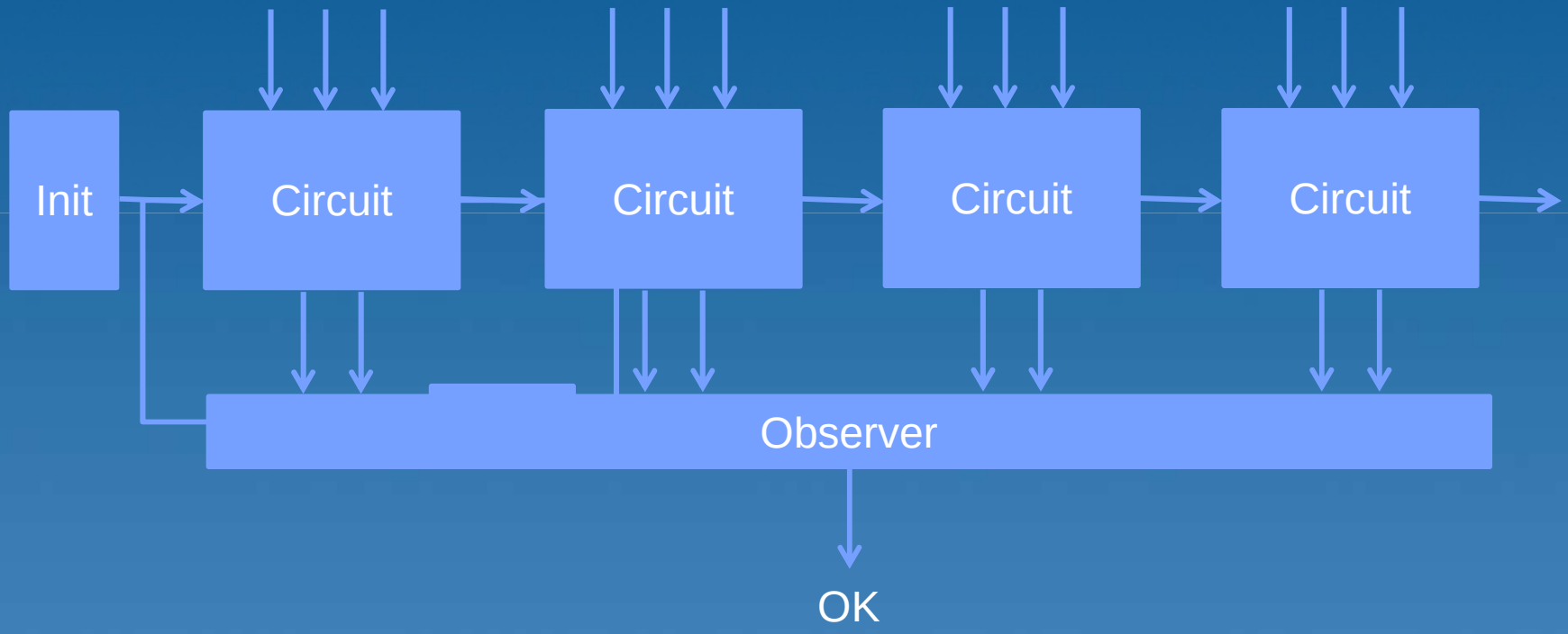
- If property possible to rewrite as a circuit:



- Reduced problem:
  - Prove that OK is always true
  - or find assignment where OK is false
  - For combinational circuit: easily done by SAT solver

Handling Sequential Circuits

Bounded Model Checking (BMC)



Bounded to unbounded

- BMC produces bounded proofs:
  - A bounded proof of depth 4 guarantees that no CEX of length 4 or shorter
  - …or finds a CEX.
- Different techniques to produce unbounded proofs:
  - Temporal induction
  - Using fixpoints

Adding proof power

- Simplifications
  - Isolating relevant parts of circuit
    - Cone of influence (COI)
  - Shrinking relevant parts
    - Verify 4 bit bus instead of 64 bis
  - Structural simplification
- Abstraction
  - Three-valued semantics
  - Automated abstraction refinement
- Proof parallelization
- Different logical systems
  - SAT (propositional logic), BDD, SMT, FOL

Three valued simulation

- Use ternary logic: {0,1,X}
  - X: don't care
- Introduce X at (some) inputs and initial flop values
- Large parts of circuit disappears
- Results:
  - OK=1: Property proven
  - OK=0: Counter example found
  - OK=X: Too many X
- The challenge is to introduce enough but not too many X

A	B	A & B
X	X	X
X	0	0
X	1	X
0	X	0
0	0	0
0	1	0
1	X	X
1	0	0
1	1	1

Abstraction refinement

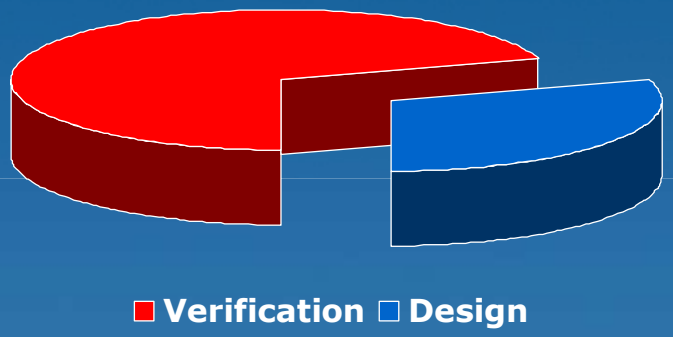
- Start with a heavily abstracted circuit
- While (proof not found)
  - Is CEX spurious (false due to X)?
    - Then analyze what X may cause this, replace it by fresh variable
    - Else report CEX
- Report valid

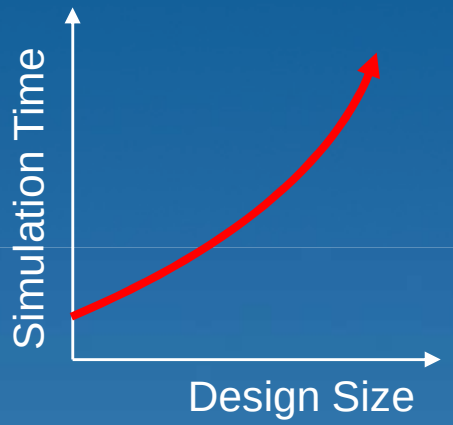
THE VERIFICATION PROBLEM

JASPER design automation

Functional Verification is a Huge, Growing Problem

- Verification consumes up to 70% of design resources





- The problem is growing exponentially
  - More simulation is not the answer

The Stakes Are High

- "A majority of ASICs/ICs require *at least* one respin. 71% of respins are due to functional bugs verification should have caught."

  - Collett International Research, Inc.

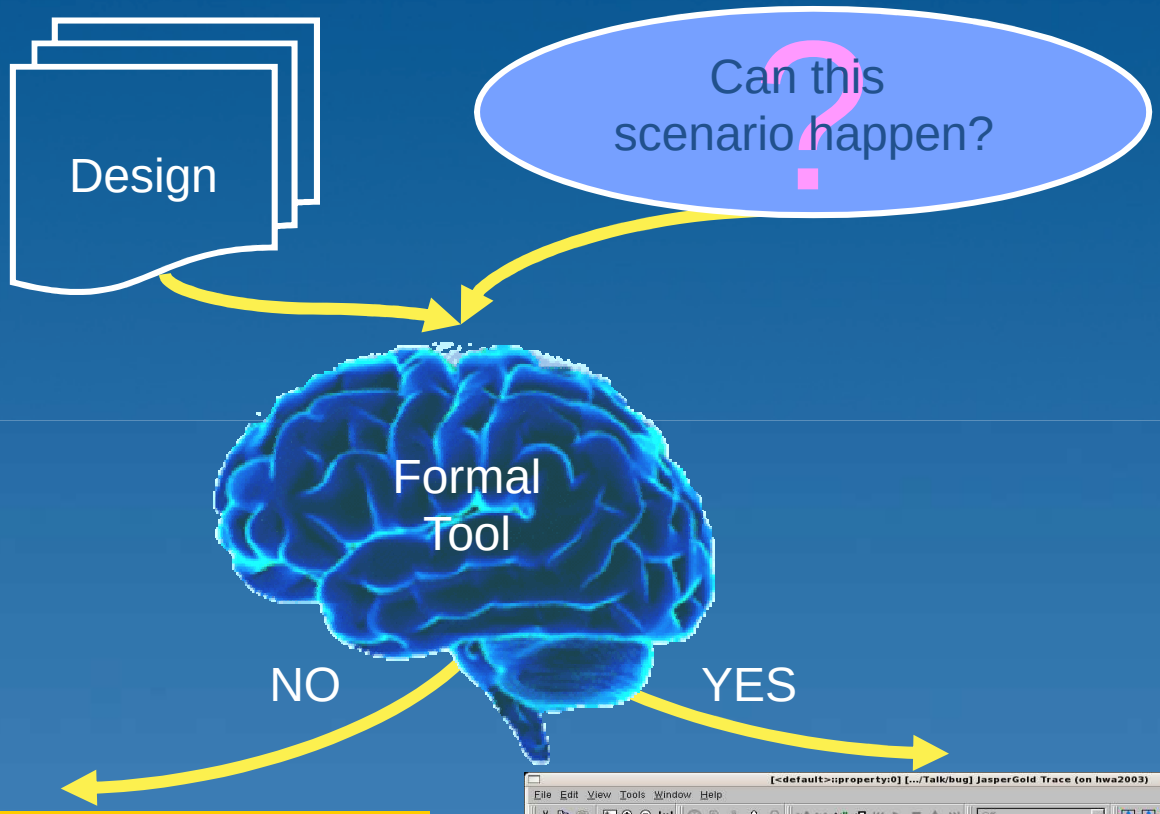
Cost of ASIC/SoC mask set = $250K to $1M+

Total cost of respin to project = ~$10M

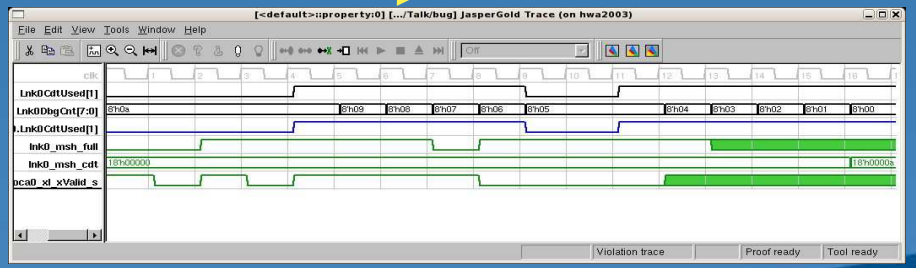
Verification Methodologies

- Dominating verification methodologies
  - Directed Simulation
    - Requires specification of test bench
    - Requires manual targeting of corner cases
  - Constrained Random Verification
    - Requires specification of stimulus constraints
- Verification quality measurements
  - Coverage metrics
    - Line coverage
    - Expression coverage
    - Functional coverage

Formal Technology: A Simple View



Guarantee that the scenario cannot happen for *ALL possible ways* design can operate



Pros and Cons of Formal

- Pros
  - Enables full proofs
  - Eliminates need for enumeration of corner cases
  - Focus on what to verify, not how to verify it
- Cons
  - Does not scale as well as simulation
  - Illegal behavior often needs to be defined (Constraints)
- Formal has a “sweet spot” where it is very valuable, but will never replace simulation.

Who Uses Formal?

- Mostly ASIC vendors
  - FPGA verification not as critical
- Proliferation
  - Still limited
  - Mostly dedicated verification people
  - Some companies have dedicated formal teams
- Trends over recent years
  - Market is clearly growing
  - Most people have now heard about formal
  - Some companies are looking for wider proliferation

WHERE TO APPLY FORMAL

JASPER design automation

Where to Apply Formal: Design Size

- "How large blocks can your tool handle?"
  - No good answer to this question!
  - Totally function dependant
  - Fundamental problem is NP complete
- Rule of thumb, Focus on:
  - Designer sized blocks
  - Critical functionalities
    - "Ensure Correctness Where it Matters Most"

Where to Apply Formal: Functionality

- Formal is not good for everything!
- Good candidates:
  - Data transportation
  - Control logic
  - Parallel interactions
- Bad candidates:
  - Data transformation
  - DSP (Digital Signal Processing)
  - Mathematics (FPU)
  - Data encryption

Good Design Candidates for Formal

- Arbiters
- On-chip bus bridge
- Power management unit
- DMA controller
- Host bus interface unit
- Scheduler, implementing multiple threads
- Virtual channels for QoS
- Interrupt controller
- Memory controller
- Token generator
- Cache coherency
- Credit manager block
- Standard interface (USB, PCI Express…)
- Proprietary interfaces
- Clock disable unit

Common characteristics of these blocks:

Concurrency and multiple data streams, which are difficult to completely verify using simulation

Example 1: Network traffic manager

- Bandwidth allocator for network switch
  - Customers buys a certain bandwidth access (eg 10 Mb/s access)
  - Switch must ensure that:
    - Customer gets at least 10 Mb/s access
    - Customer does not get more that 10 Mb/s access
  - Each customer can buy different bandwidth sizes
    - 256 Kb/s
    - 512 kb/s
    - ...
    - 10 Mb/s
    - ...

Example 1: Network traffic manager

- Bandwidth allocation controlled by credit manager
  - Buying a bandwidth of speed n gives you x credit tokens on the switch
  - The tokens denote access to switch memory
  - Packet enters design: 1 token deducted from credit pool
  - Packet exits design: 1 token returned to credit pool
- Verification problem
  - Are token always returned correctly?
  - Failing to do so could cause token leakage
  - Memory access would be blocked
  - Switch would hang

Example 1: Network traffic manager

- Problem type: Token leakage verification
- Problem characteristics
  - Huge number of possible scenarios
  - Hundreds of communication channels active at the same time
  - Impossible to verify sufficiently with simulation
  - Corner case bug could make switch unusable
    - 1 token leaked every second would force reboots every day
- Perfect fit for formal
  - Impossible to enumerate corner case scenarios
  - Full proof important

Example 2: Microcontroller

- Microcontroller supporting two simultaneous execution threads
- Verification Problem:
  - Does instruction execution behave according to spec?
- Property example:
  - Instructions in memory should be executed sequentially
- Problem characteristics:
  - Huge number of possible scenarios
    - Combinations of instructions
    - Thread context switching
    - Interrupt handling

Example 2: Microcontroller

- Flow control bug found
- Condition:
  - Both threads active
  - Thread 1 executes branch
  - User interrupt kills thread 1 at the same cycle as branch instruction executes
- Symptom:
  - Branch information not cleared
  - Causes Thread 2 to branch instead
- Bug characteristics:
  - Requires a very specific and cycle accurate scenario to occur
  - Almost impossible to find with simulation

FORMAL VERIFICATION CHALLENGES

JASPER design automation

What Makes a Property Hard to Prove?

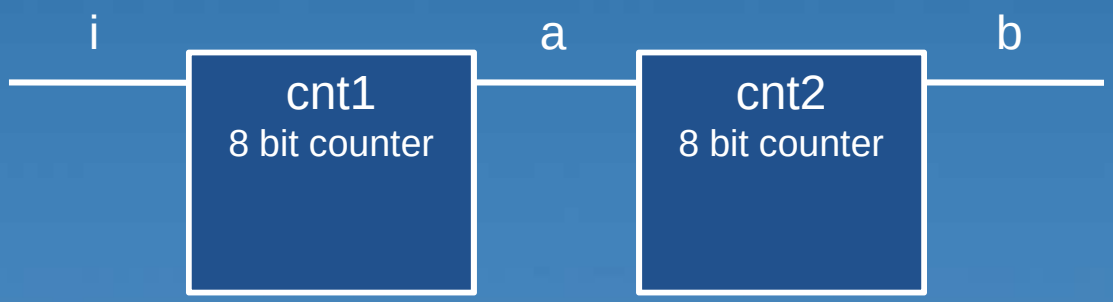
- Example:
  - A memory has an 8 bit wide data bus and an 8 bit wide address bus.
  - **Property:** If you write data to an address, then the next time you read from that address you should get the same value back as you wrote in unless you have performed another write in the mean time.
- How would this be verified in simulation?
- Why is this problem hard to prove?

State Space Complexity

- The State Space problem
  - Formal verification explores all possible states
- What is the size of the state space of the previous design?
  - Word size is 8 bits
  - 8 bit wide address means 2^8 words.
  - Total number of memory bits: 8*2^8 = 2048 bits
- What is the total number of distinct states that the memory can be in?
  - 2^2048 = 3.32 * 10^616
  - Estimated number of atoms in the observable universe: 10^80

What Makes a Property Hard to Prove?

- Example:
  - Functionality:
    - An 8 bit counter, “cnt1”, counts the number of times an input signal has been high.
    - Signal “a” is high when “cnt1” is full.
    - An 8 bit counter, “cnt2”, counts the number of times “a” has gone high.
    - Signal “b” is high when “cnt2” is full.
  - Property:
    - “a” and “b” are never active at the same time.



What Makes a Property Hard to Prove?

- Why is it hard to find a counter example for this problem?
  - Number of memory bits are just 2*8
  - State space is not a big problem

Sequential Depth Complexity

- How many reachable states are there at any given distance from reset?
  - 1 cycle: cnt2 = 0 and cnt1 = 0 or 1 - #states: 2
  - 2 cycles: cnt2 = 0 and cnt1 = 0,1 or 2 - #states: 3
  - 3 cycles: cnt2 = 0 and cnt1 = 0,1,2 or 3 - #states: 4
  - ...
  - 256 cycles: cnt1 = 0 to 256 and cnt2 = 0 or cnt1 = 0 and cnt2 = 1 - #states: 257
  - ...
  - 65535 cycles: cnt1 = 0 to 256, cnt2 = 0 to 256 - #states: 65536
- JasperGold has to verify all of the 65535 steps before finding a CEX!

Engines and Design Complexity

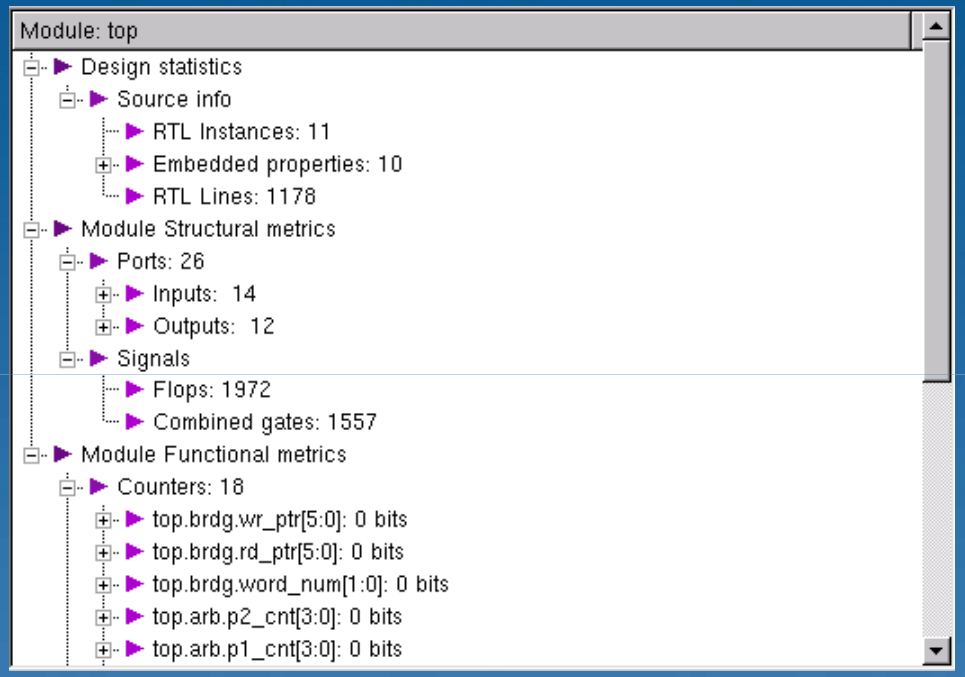
- Main reasons for performance problems:
  - State Space Size
  - Sequential Depth
- Proof engines do not use brute force to verify all combinations
  - Doing so would cause most problems to blow up
  - The different engines use different algorithms to handle verification problems efficiently
- Different engines have different strengths and weaknesses
  - Formal Expert contains information about engine selection.

Recognizing a Hard-to-Prove Problem

- Worst case scenario reasoning
  - What is the longest possible trace I would get if there is a bug in my design?
- Example:
  - **Property:** Data integrity across a bus bridge
  - *What if:* Data is corrupted when my FIFO underflows?
    - Underflow can happen at cycle 2, bug can be detected around cycle 2.
  - *What if:* Data is corrupted when my FIFO overflows?
    - Overflow can not happen until at least after FIFO length number of operations. Bug can only be detected after that. Investigate how large the FIFO is!

Formal Predictor Improves Verification Predictability

- Identifies complex logic before formal analysis
- Provides a detailed report on the design's complexity
- Enables user to decide where to safely apply abstractions to improve verification performance
- Multiple views
  - Analysis Region
  - Cone of Influence
  - Full Design

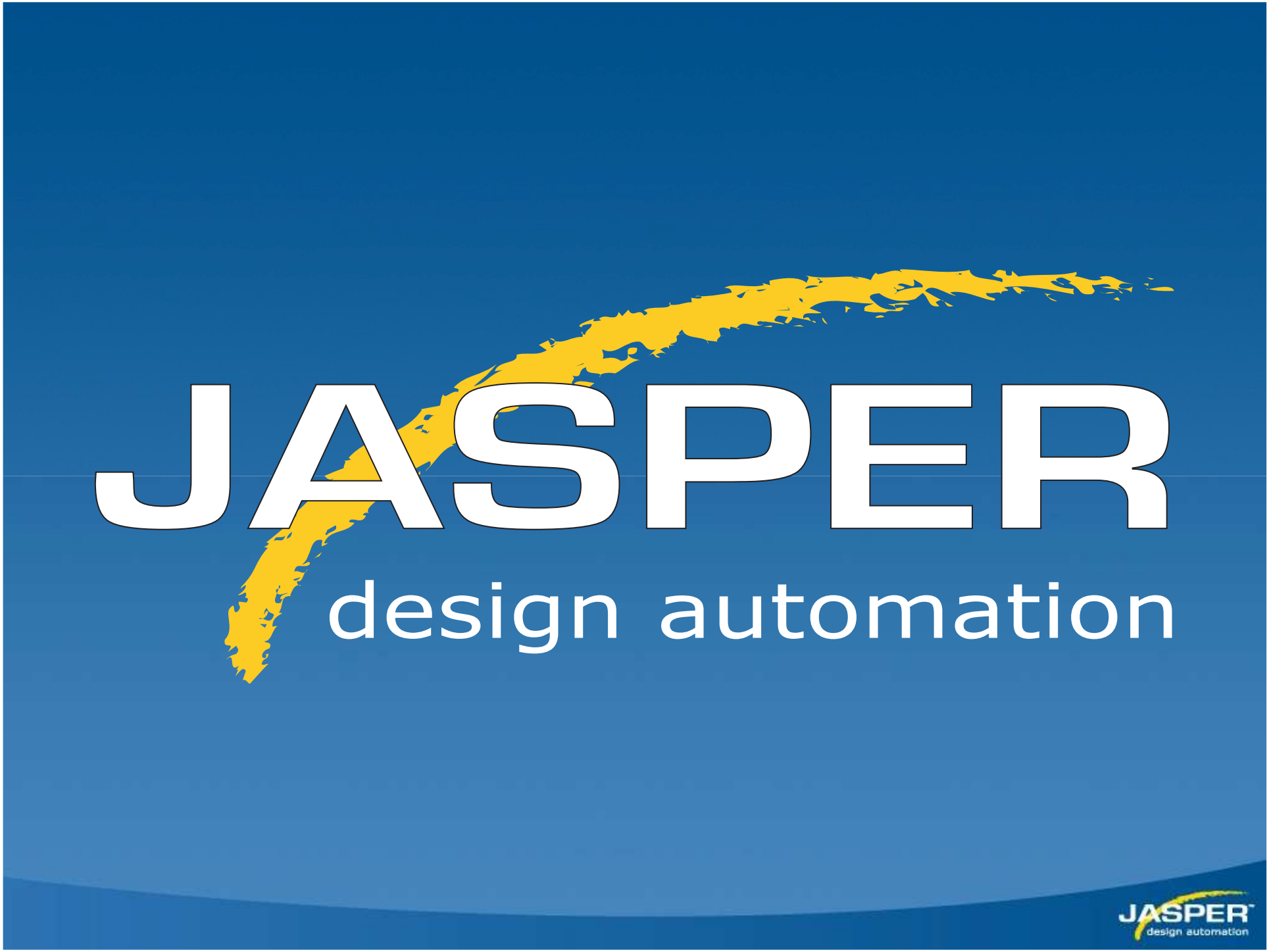


Coping with Formal Complexity

- Methodology
  - Appropriate size design blocks to apply formal analysis on
  - Formal friendly modeling of properties and constraints
  - Leverage symmetries in the design
  - Assume/guarantee reasoning
- Technology
  - Safe abstraction techniques
  - High performance engines

Formal Testplan

- Hierarchical definition of design functionality
- Identify functional areas:
  - What functionality is the design supposed to deliver?
- Example: PCI network card
  - Interface protocol compliance
    - Standard protocol rules must be respected
  - End to end data integrity
    - Packets must never be dropped, duplicated, corrupted or reordered
  - Error correction
- Gradually refine functionalities until function can be captured by a property.
  - Example: Address must remain stable during request


JASPER
design automation
JASPER
design automation