JasperGold™
High-Level Formal Verification

Vigyan Singhal
Harry D. Foster

Agenda

• Jasper introduction
• Model checking
• Block-level verification
  - High-level requirements
  - Formal testplan
  - Coverage
• Formal Testplanner
• PSL (Property Specification Language)
Jasper Design Automation

• Jasper is the premier electronic design automation (EDA) supplier of high-level formal functional verification software.

• Jasper's solution achieves 100% Actual Coverage – improving the quality of electronic design, predictably and within verification schedule constraints

• Jasper has unique and valuable technology that changes the verification game and “makes formal verification real”.

Jasper Design Automation

• Founded in 1999 (originally Tempus Fugit)
• Over $20M in funding -- led by Foundation Capital, Accel Partners
• Management
  - Kathryn Kranen  President and CEO (Verisity, Quickturn)
  - Vigyan Singhal  CTO (Cadence, UC Berkeley)
  - Harry Foster  Chief Methodologist (Verplex, HP)
  - Craig Cochran  VP Marketing (Synopsys)
  - Craig Shirley  VP Sales (Verisity)
  - Nafees Qureshy  VP Engineering (CoWare)
• 9 PhDs in Formal Verification
  - UC Berkeley, Stanford, CalTech, Chalmers, Gothenburg
Sequential Satisfiability (model checking)

Is there a sequence of input assignments such that p is 1 at any finite time?

Combinational gates + flops (with initial values)
Sequential Satisfiability (model checking)

Is there a sequence of input assignments such that \( p \) is 1 at any finite time?

\[
\begin{array}{c}
a \quad 1_0 \\
\downarrow & \\
\quad 1_0 & 1_0 \\
\quad \downarrow & \\
\quad 0_0 & 0_0 \\
\quad \downarrow & \\
\quad 0_0 & 0_0 \\
\quad \downarrow & \\
\quad p \\
\end{array}
\]

Sequential Satisfiability (model checking)

Is there a sequence of input assignments such that \( p \) is 1 at any finite time?

\[
\begin{array}{c}
a \\
\downarrow & \\
\quad 1_0 \\
\quad \downarrow & \\
\quad 0_0 \\
\quad \downarrow & \\
\quad 0_1 \\
\downarrow & \\
\quad q \\
\end{array}
\]
Sequential Satisfiability (model checking)

Is there a sequence of input assignments such that $p$ is 1 at any finite time?

Sequential Satisfiability (model checking)

Is there a sequence of input assignments such that $p$ is 1 at any finite time?
Sequential Satisfiability (model checking)

Is there a sequence of input assignments such that

\[ p \text{ is 1 at any finite time} \]

Combinational gates + flops (with initial values)
Complexity: PSPACE-hard (Aziz 93)

Model Checking use model

- Does the design have a given desirable property?

Properties and Constraints

Design under test

- Properties - Specification
- Constraints - Assumptions needed to prove Properties
- Design under test - Implementation
Creating the chip-level testbench can be as difficult as creating the chip model.

- **Poor coverage**
  - Full chip verification does not provide enough coverage to find all bugs.
  - High controllability of block-level interfaces is difficult to achieve from stimulus generated at the chip boundary.

- **Low observability**
  - Internal errors during full chip simulation might not propagate to an observable output during the test.

Most of today's verification is performed at the chip level. Very little block-level verification is performed.

**Cost of a re-spin > $1M + months of delay**
What is necessary for FV to succeed in industry?

FV solution must provide:

1. High Return-on-Effort
   - Measurable proof of quality
2. Ease of use (FV expert not require for successful)
   - Encapsulate formal expert knowledge into the tool
   - Strong engines + advance abstraction techniques
   - Guide the user to success
3. Sophisticated debug capabilities
   - Design illumination—enabling user to learn the design
4. Means to overcome the "blank page" syndrome
   - Specification challenge
   - Methodology challenge
5. Metrics to measure progress and success

What’s the solution?

- Give up, accept the raw algorithm limitations and perform bounded-model checking, or....

- How do we handle complexity for other applications?
  - For example, place-n-route is an NP-complete problem.
  - In place-n-route, using guidance, the tool can automatically partition the problem down into tractable pieces.

- To succeed, formal verification tools must combine advanced techniques with multiple, powerful search engines
  - For example, managed automatic tunneling, ....
  - Encapsulate the knowledge of the formal expert into the tool!
JasperGold Manages Proof Capacity  
_Tunneling, Monitors, and Abstraction_  

**High-level requirement**

- Conventional and Hybrid Formal Tools Choke on All the Logic in the Cone of Influence.  
- Progress and Coverage are Unknown.  

JasperGold Refines the Proof Space  
Abstrasts Formal Unfriendly Structures and Displays Progress and Coverage  

**Inputs**  
**Outputs**  

---  

JasperGold - Ease of Use

- Knowledge base of design-specific methodology and requirements templates
- Interactive, context-sensitive debugging environment

- **High-Level requirements**
  - Static, exhaustive proofs without simulation
  - Proof navigation suggests next steps
  - Isolates the root cause of each bug to the faulty line of RTL

- **RTL Design Block**
  - Functional Bugs Isolated
  - Proven 100% Correct
In-line assertions (PSL example)

```verilog
module btrack (clk, allocate, deallocate, set);

input clk, allocate, set, deallocate;
reg active_r, valid_r;
wire active_s, valid_s, active_in, valid_in;

assign active_in = ((allocate | active_s) & ~deallocate);
always @(negedge clk)
    active_r <= active_in | set;
assign active_s = active_r;
assign valid_in = (active_s & (set | valid_s));
always @(negedge clk) begin
    valid_r <= valid_in;
    assign valid_s = valid_r;
    // psl property deallocate_when_not_active =
    //     always {deallocate} |-> {~active_s; valid_r};
    // psl assert deallocate_when_not_active;
endmodule
```

A high-level requirement example

[DesignCon 04 PCI Express Paper]

High-level requirement = no packets are dropped, duplicated, or corrupted

- Implementation assertions: low-level assertions
  (similar to writing simulation monitors)
Requirement: A transaction from an AHB Master on an external device must NOT get duplicated, corrupted or dropped, when it goes out to the Wishbone fabric.

Illustrating with an Example

Illustrating the packet ordering requirement using a FIFO
HLR example

always @(posedge clk or posedge rst)
if (rst) begin
  wrPtr <= 4’d0; rdPtr <= 4’d0;
  for (i = 0; i < 4; i = i + 1)
    stored[i] = 64’b0;
end else begin
  if (datain) begin
    wrPtr <= wrPtr + 4’d1;
    store[wrPtr] <= din;
  end
  if (dataout) rdPtr <= rdPtr + 4’d1;
end

prove { ~ ((dataout) && (dout != store[rdPtr])) }
prove { ~ (((wrPtr + 4’d1) == rdPtr) && ~dataout && datain) }
prove { ~ ((wrPtr == rdPtr) && (dataout)) }

Notice dependency on number of transactions in flight

High-level Requirements: high return on effort

High-level Requirements
- End-to-end
- Black box
- Based on design intent
- Are harder to prove

Low-level Assertions
- Localized
- Implementation-specific
- Can find bugs faster

Data Integrity
Arbitration Fairness
Intent
Implementation

Design Behavior

One Hot
Increment By 1
FIFO Overflow
FIFO Overflow

Jasper design automation
High-Level Formal vs. Conventional Formal

JasperGold operates here

μ-architecture Specification → High-Level Requirements
RTL Coding → Implementation Assertions
High-Level Requirements

Intent
Implementation

Most formal tools operate here

Seven steps of a formal testplan

1. Identify good candidates
2. Identify
3. Define interface
4. Requirements checklist
5. Formal description
6. Verification strategy
7. Coverage goals

assert_never ();
Formal Testplan Example

- Transmit Path (tx)
  - Packets should not lose ordering or get dropped
  - Packets should not get duplicated
  - Packets headers should not get corrupted
  - Link Credits sent are less than or equal to Network Credits received
  - No Link Credits received from the network are lost (every credit is sent downstream, or recorded locally)
  - No packets are retried until timer expires
  - Any packet not ack’ed is retried
  - Retried packets should not lose ordering or get dropped
  - Retried packets should not get duplicated

- Receive Path (rx)
  - Alignment
  - Packet Latency
  - …

Overcoming the formal specification “blank-page” syndrome
JasperGold Proving Environment - Sophisticated Debug

- Design Illumination mode steers the user through the design - "lets you get into the designer’s head without talking to them"
- Enables you to quickly expose the root cause of a bug, isolated down to the faulty line of RTL code
- Provides design-specific next steps and proof progress metrics

Are there success stories?

- **Real** examples include:
  - Arbiters of many different kinds
  - On-chip bus bridge
  - Power management unit
  - DMA controller
  - Host bus interface unit
  - Scheduler, implementing multiple virtual channels for QoS
  - Interrupt controller
  - Memory controller
  - Token generator
  - Credit manager block
  - Standard interface (AMBA, PCI Express, …)
  - Proprietary interfaces
  - Clock disable unit

- Common characteristic: *concurrency, multiple data streams*

Multiple, concurrent streams
Hard to completely verify using simulation
"10 bugs per 1000 gates"

-Ted Scardamalia, IBM
Control, data transport

- Concurrent
- Handles multiple streams
- “Corner-case” bug is missed because randoms never excited that timing combination
- Bug example: during the first 3 cycles of a transaction start from one side of the interface, a second transaction came in on the other side of the interface and changed the config register.
- Design examples: arbiters, standard interface protocols (PCI), DMA controller

Data transform

- Sequential, functional
- Single-stream
- “Corner-case” bug is missed because testplan was not complete (spec was large)
- Bug example: the IFFT result is incorrect if both inputs are negative
- Designs: floating point unit, graphics shading unit, convolution unit in a DSP chip, MPEG decoder
On-chip Bridge (e.g. AHB-AHB)

- HLRs
  - Bridge transfers data correctly from one end to the other (accounting for 16-bit to 64-bit data width change, and splitting of long burst packets)
  - Bridge is fair to both masters
- Complexity
  - Timing relationship between the two masters
  - Error responses from the slave with arbitrary timing relationship to what is happening in the master
- Corner-case
  - Bridge locks up if slave issues error on 2nd cycle of a 4-beat request from master A, and master B makes a new request on the same cycle the error is passed on to master A

Verification solution must reduce collateral damage

- Today, design concurrency and complexity are explored late in the flow
- The problem is that at this state massive design interdependencies exist fully-developed RTL
- Late stage bugs can have significant collateral damage
JasperGold Delivers “*Provably* Correct Design”

Incidentally add design functionality, prove its correctness, and then ensure that it remains correct through final regressions.

Two Reasons for Formal Verification

"By far, we [at Intel] have found that the greatest value of formal verification is assurance" – John O'Leary, Intel

**Full proof** provides the most value
Most of today's verification is performed at the chip level. Very little block-level verification is performed.

Cost of a re-spin > $1M + months of delay

Design and Verification Flow

Design Cycle Comparison

Simulation Flow

Simulation + Formal Verification Flow

Design and formal verification performed in parallel - 100% Proof
Retrun on Investment

Conclusion: Making Formal Verification Mainstream

- Higher returns
  - Block verification on real-sized designs
  - Expectation and predictability of results
  - For even higher results:
    - Interactivity and methodology support
    - User creativity always adds more
- Higher investment from the user
  - Lower investment \rightarrow marginal payoff
  - Cannot be factored in project schedules
  - Need predictable effort estimates
  - User interaction is acceptable, if effort can be quantified and estimated up front
  - Today, simulation can be replaced: launch-and-forget will not replace simulation
  - Don’t short-change formal by applying only to local assertions