Deploying formal in a simulation world

VIGYAN SINGHAL
OSKI TECHNOLOGY
Formal from different vantage points

University Researcher
(4 yrs 1991-1995)
(UC Berkeley)
Goal: advance state-of-the-art

Semiconductor tool user
(6 yrs 2005-2011)
(Oski)
Goal: optimize $ and time-to-market

EDA tool developer
(10 yrs 1995-2005)
(Cadence, Jasper)
Goal: build competitive tools
Goal: advance state-of-the-art

Areas of concern

- Temporal logics (CTL, CTL*, PLTL)
- Fairness and $\omega$-automata
- Complexity
  - Known: CTL Model checking linear time complexity (size of FSM)
  - Proved: CTL model checking PSPACE-complete (size of design)

- Time to returns: almost infinite

- Anecdote: “model check a 9-state FSM at Motorola”
Goal: build competitive tools

Areas of concern

- Verilog/SystemVerilog parsing
- User interface and GUI
- Property synthesis: PSL and SVA

Time to returns: 4-5 years

Anecdote: “lost an eval at Intel because tool ran for 28 days”
Goal: optimize $ and time-to-market

Areas of concern

Verification planning

Metrics to measure progress, and when we are done

Integrate simulation and formal planning and results

Abstraction (and reductions) are key to making formal productive

Time to returns: 3-6 months

Anecdote: “how did you miss a bug on a formally verified block?”
Types of post-silicon flaws

Verification is still the largest problem

Verification market size (2009)*

* excluding analog

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Formal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4M</td>
<td>$401.8M</td>
</tr>
<tr>
<td>0</td>
<td>0.4M</td>
</tr>
</tbody>
</table>

- **Gate-level formal (equivalence checking)**
  - Then (1993): Chrysalis; **Now**: Cadence (Verplex), Synopsys

- **RTL formal (model checking)**
  - Then (1994): Averant, IBM; **Now**: Jasper, Mentor (0-In)

Source: Gary Smith EDA, October 2010
Formal tool usage in industry

- Around for 20 years
- Expectations has been set high
  - Low efforts for constraints
  - Tools run fast enough
- Expectations have been set low
  - Only verify local assertions
  - No End-to-End proofs
- Perception: low ₩/$

- Training and staffing
  - Few places to learn formal application
  - Single user should not do both formal and simulation

Source: xkcd.com
Biggest challenges needing solutions

- Verification management and coverage

MANAGING THE VERIFICATION PROCESS

DEFINING APPROPRIATE COVERAGE METRICS

KNOWING MY VERIFICATION COVERAGE

CREATING SUFFICIENT TESTS TO VERIFY THE DESIGN

Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission.
Achieving verification closure

Plan
- Partition Verification between Formal and Simulation

Verify
- Apply Abstractions for Verification Convergence

Measure
- Integrate Formal and Simulation Coverage
Where to apply model checking

“Control”, “Data Transport” designs

- Arbiters of many kinds
- Interrupt controller
- Power management unit
- Credit manager block
- Tag generator
- Schedulers
- Bus bridge
- Memory Controller
- DMA controller
- Host bus interface
- Standard interfaces (PCI Express, USB)
- Clock disable unit

Multiple, concurrent streams
Hard to completely verify using simulation

“10 bugs per 1000 gates”
-Ted Scardamalia, IBM
“Data transform” designs

- Floating point unit
- Graphics shading unit
- Inverse quantization
- Convolution unit in a DSP chip
- MPEG decoder
- Classification search algorithm
- Instruction decode

Where not to apply model checking

Single, sequential functional streams
“2 bugs per 1000 gates”
-Ted Scardamalia, IBM
Formal (MC, SEC*) and simulation strengths

* SEC = Sequential Equivalence Checking (RTL vs C model)
How perfect does formal have to be?

- Not all bugs need to found/fixed
- Formal does not need to find the last bug
- Usually bounded proofs are good enough
  (if bound is good enough!)
- Formal has to be more cost-effective than the alternative
Bug-fix cost rises exponentially

- Block-level design
- Block-level verification
- Chip-level verification
- ECO phase

Tapeout
Silicon is back
Verification manager’s dashboard

Coverage tracking

Bug tracking

Runtime status
A simulation testbench

Design Under Test (DUT)

Bus Functional Models (BFM) (Input stimulus generator)

Checkers (Scoreboard)

Test #1
Test #2
Test #N

Coverage (code and functional)
A formal testbench

Design Under Test (DUT)

Constraints

Coverage (code and functional)

Checkers (Scoreboard)

Abstraction Models
• Checkers
• Constraints
• Complexity
  • (using Abstraction Models)

• … and Coverage (to measure completeness of formal)
Traditional formal verification

- Usually based on Local Checkers:
  1. RTL assertions
  2. Interface assertions
- Useful for bug hunting
  - Not for finding all/most bugs, or as replacement for simulation effort
- For replacing simulation, need End-to-End Checkers
- Run into complexity barrier
  - For medium- or large-sized designs, run into state space explosion
  - Without Abstraction Models, cannot scale complexity
Checkers (End-to-End)

- For End-to-End formal verification, less than 5% of Checker code is SVA; rest is SV or Verilog
  - (Synthesizable) Reference model is typically as big an effort as the RTL
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
if (~rst) st <= 2'b00;
else case( st )
    2'b00: if (~a) st <= 2'b01;
    2'b01: st <= 2'b10;
    2'b10: if (a) st <= 2'b00;
endcase

always @(posedge clk or negedge rst)
if (~rst) b <= 1'b0;
else if (~a | b) b <= 1'b0;
else b <= 1'b1;

Checker: (st == 2'b01) => ~b

RTL

Internal Netlist

a

Internal STG

st[0]

st[1]

b

\[2^3 = 8\]

\[2^{10} = 1,024\]

\[2^{20} = 1,048,576\]

\[2^{30} = 1,073,741,824\]
Complexity – function of Cone-of-Influence

- One coarse measure of Complexity
  - number of flops/memory bits in the Cone-of-Influence of the Checker
State space complexity
**Abstractions (to manage complexity)**

- An “Abstraction” of a design is a design that has a superset of the design behavior
- Useful to overcome complexity barriers
  - Smaller Cone-of-Influence
  - Shallower search space
  - Ability to skip long initialization sequences
- Cannot give a false positive
- Can give a false negative (Fail), but…
  - You get a trace to determine the reason for the negative
Complexity (and Abstractions)

Effect of abstractions:
- Reduces state space
- Adds state transitions
- Adds Reset states
Overcoming complexity with Abstractions

Without Abstractions

With Abstractions

Realistic design sizes
Examples of Abstractions

- Allow DUT to reset to a deep state
  - BANKS_IDLE state for a memory controllers (skips thousands of clocks of initialization)
- Replace memory by a memory model tracking a specific Byte of a specific Beat of a specific Transaction
  - 13\textsuperscript{th} transaction, 243\textsuperscript{rd} beat, 1\textsuperscript{st} Byte
- Replace a Tag Generator by an abstract model
  - Reduce sequential depth by tracking specific value
  - Example in the next few slides…
Example: PCIe Transaction Layer

- 128-bit datapath
- 8 VCs
- History, policy-dependent DRR scheduler
- Conf responses arbitrated with TLPs and FC DLLPs
- Tx, Rx Buffers can store multiple TLPs (upto 32kb each)
Abstraction for Tag Allocator

- Pick an arbitrary, but fixed tag: e.g. tag #79
- Replace Tag Allocator by a two-state Abstraction:
  - HAS_79 (H): models that Tag_Allocator has tag #79
  - DOESNOT_HAVE_79 (D): Tag Allocator does not have #79
**Abstraction for Tag Allocator**

A. Use Tag Allocator Abstraction (DUT = Design with abstracted TA)
   - Add constraints
     1. (state == H) |-> (!empty);
     2. (state == D) |-> (tag_out != 79);
   - Add assertions:
     1. (state == H) |-> (tag_return != 79);
     2. (state == D) |-> Tag #79 is eventually returned

B. Prove Tag Allocator Abstraction (DUT = Tag Allocator RTL)
   - Reverse constraints and assertions (e.g. prove no tags leak)
   - Can be a sequentially long, but on a tiny DUT
Abstraction Model

• Other example of Abstraction Models:
  • Localization
  • Datapath
  • Memory
  • Sequence
  • Counter
  • Floating pulse

• Without Abstraction Models:
  • On most interesting designs, formal tools do not search far enough
Verification closure with formal and simulation

OSKI TECHNOLOGY, INC.

Coverage on RTL designs

RTL (Verilog)

1. reg p;
2. always @(*) begin
3.   if (a || (b && c))
4.     p = d;
5. else
6.     p = e;
7. end

Equivalent RTL

1. reg w, p;
2. always @(*) begin
3.   w = a || (b && c);
4. end
5. always @ (*) begin
6.   p = (w && d) || (!w) && e;
7. end

Synthesis

Gate-level netlist

b

<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
</tr>
<tr>
<td>---</td>
</tr>
</tbody>
</table>

p
d

e
### Input Coverage: line/expression coverage

```verbatim
1. reg p;
2. always @(*) begin
3.   if (a || (b && c))
4.     p = d;
5.   else
6.     p = e;
7. end
```

<table>
<thead>
<tr>
<th>Line coverage</th>
<th>Expression coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Target #1: (a || (b && c))
Target #2: (a || (b && c))

- **Line coverage**
- **Expression coverage**
- **Target #1**: (a || (b && c))
- **Target #2**: (a || (b && c))

#### Notes
- Input Coverage: line/expression coverage
- Target #1:
- Target #2:
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
if (~rst) st <= 2'b00;
else case( st )
  2'b00: if (~a) st <= 2'b01;
  2'b01: st <= 2'b10;
  2'b10: if (a) st <= 2'b00;
endcase

always @(posedge clk or negedge rst)
if (~rst) b <= 1'b0;
else if (~a | b) b <= 1'b0;
else b <= 1'b1;
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
if (~rst) st <= 2'b00;
else case( st )
  2'b00: if (~a) st <= 2'b01;
  2'b01: st <= 2'b10;
  2'b10: if (a) st <= 2'b00;
endcase

always @(posedge clk or negedge rst)
if (~rst) b <= 1'b0;
else if (~a | b) b <= 1'b0;
else b <= 1'b1;
Coverage-driven simulation methodology

- RTL
- Spec
- Verification Plan
- BFM
- Checker
- Coverage Model
- Tests
- Waiver List
- Coverage Analysis

Flow:
- RTL to Spec
- Spec to Verification Plan
- Verification Plan to BFM
- BFM to Checker
- Checker to Coverage Model
- Coverage Model to Tests
- Tests to Coverage Analysis
- Coverage Analysis to Waiver List
- Waiver List to RTL

Additional flows:
- Tests to More tests
- Constraint and bias refinement to BFM

Notes:
- More tests Feedback loop
- Constraint and bias refinement for BFM improvement
Coverage for hardware designs

- Trivial to get to 60-70% code coverage
- 100% line/expression coverage often required for tapeouts
  - Manual waivers are allowed
- NVIDIA SNUG 2011 paper
  - 270 man weeks to do waiver analysis for one design
  - 180 man weeks to write missing tests
Coverage database collection

Testlist #1

AXI BFM

Transaction Layer

Data Link Layer

Physical Layer

PCIe BFM

AXI-PCIe bridge checker

Coverage

Coverage

Coverage

Coverage DB #1
Coverage database collection

- Testlist #2
- AXI BFM
- Transaction Layer
- Data Link Layer
- Physical Layer
- PCIe BFM
- AXI-PCIe bridge checker
- Coverage
- Coverage
- Coverage
- Coverage DB #1
- Coverage DB #2
Coverage is not the be-all and end-all

“\textit{The perfect is the enemy of good}”
\hfill \textit{-Voltaire (1772)}

- Coverage is not perfect
  - Bugs are missed even with 100\% coverage
- But…
  - Helps measure progress
  - Helps identify blind-spots
Input vs Observable coverage

• “Have I verified enough input sequences” (Input coverage)

• “Is my set of checkers complete enough” (Observable coverage)

• Same two notions apply for both simulation AND formal

• Bounded model checking (BMC) is the most used formal technique

NOTE: Formal does not verify all possible input sequences
### Coverage Summary Report, Instance-Based

#### Top Level Summary

<table>
<thead>
<tr>
<th>Instance name: mic</th>
<th>Module/Entity name: mic</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Total</th>
<th>Block</th>
<th>Expression</th>
<th>Toggle</th>
<th>FSM</th>
<th>Assertion</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>82%</td>
<td>95%</td>
<td>100% (3/3)</td>
<td>96% (292/3056)</td>
<td>100% (24/24)</td>
<td>20% (1/5)</td>
<td>Cumulative</td>
</tr>
<tr>
<td>97%</td>
<td>No items</td>
<td>No Items</td>
<td>97% (412/424)</td>
<td>No Items</td>
<td>No Items</td>
<td>Self</td>
</tr>
</tbody>
</table>

#### Coverage of immediate sub-instances:

<table>
<thead>
<tr>
<th>Total</th>
<th>Block</th>
<th>Expression</th>
<th>Toggle</th>
<th>FSM</th>
<th>Assertion</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>98%</td>
<td>100% (73/73)</td>
<td>No Items</td>
<td>96% (2032/2121)</td>
<td>No Items</td>
<td>No Items</td>
<td>mic_fifo_0</td>
</tr>
<tr>
<td>58%</td>
<td>91% (29/32)</td>
<td>No Items</td>
<td>82% (52/63)</td>
<td>No Items</td>
<td>0% (0/2)</td>
<td>mic_arb_0</td>
</tr>
<tr>
<td>75%</td>
<td>96% (50/52)</td>
<td>100% (3/3)</td>
<td>79% (19/24)</td>
<td>100% (24/24)</td>
<td>0% (0/2)</td>
<td>fifo_state_0</td>
</tr>
<tr>
<td>85%</td>
<td>70% (7/10)</td>
<td>No Items</td>
<td>100% (264/264)</td>
<td>No Items</td>
<td>No Items</td>
<td>mux8_0</td>
</tr>
<tr>
<td>97%</td>
<td>100% (8/8)</td>
<td>No Items</td>
<td>92% (100/109)</td>
<td>No Items</td>
<td>100% (1/1)</td>
<td>memclt_0</td>
</tr>
<tr>
<td>99%</td>
<td>100% (5/5)</td>
<td>No Items</td>
<td>98% (50/51)</td>
<td>No Items</td>
<td>No Items</td>
<td>sram_0</td>
</tr>
</tbody>
</table>
Coverage reporting
Is my formal complete?

- Are my Checkers complete?
- Are my Constraints complete?
- Is my Complexity strategy complete?
input a;
reg b;
reg [1:0] st;

always @(posedge clk or negedge rst)
  if (~rst) st <= 2'b00;
  else case( st )
    2'b00: if (~a) st <= 2'b01;
    2'b01: st <= 2'b10;
    2'b10: if (a) st <= 2'b00;
  endcase

always @(posedge clk or negedge rst)
  if (~rst) b <= 1'b0;
  else if (~a | b) b <= 1'b0;
  else b <= 1'b1;
Input Coverage for formal

- Constraints: Environment may be over-constrained
  - Intentional: avoided some hard to model or verify input combinations
  - Unintentional: bugs in constraints; forgot to remove intentional over-constraints

- Complexity: All checkers are verified up to proof depth N
  - Any target, not reachable in N clocks, is not covered

- Checkers: does not verify completeness of Checkers
  - No different than simulation!
Coverage database collection

AXI BFM

Testlist #2

Transaction Layer

Data Link Layer

Physical Layer

PCle BFM

AXI-PCIe bridge checker

Coverage

Coverage

Coverage

Coverage DB #1 + Coverage DB #2
Formal coverage integrated with simulation

- AXI asserts
- AXI constraints
- End-to-end checkers
- Internal asserts
- Transaction Layer
- Coverage
- Coverage DB #1
- Coverage DB #2
- Formal Coverage DB
- TL constraints

Formal coverage integrated with simulation

Coverage DB #1 + Coverage DB #2 + Formal Coverage DB
Formal code coverage methodology

1. Implement Checkers and Constraints
2. Run formal verification and collect Coverage
3. Add Abstractions and/or fix Constraints
4. Are Coverage goals met?
5. Design is formally verified
Observable Coverage (using mutations)

1. reg p;
2. always @(*) begin
3.   if (a || (b && c))
4.     p = d;
5.   else
6.     p = e;
7. end

Mutant for line#4

1. reg p;
2. always @(*) begin
3.   if (a || (b && c))
4.     p = 1'bX;
5.   else
6.     p = e;
7. end

Mutant for line#6
Observable Coverage for formal

1. reg p;
2. always @(*) begin
3.   if (a || (b && c))
4.     p = d;
5.   else
6.     p = e;
7. end

Mutant for line#4

1. reg p;
2. always @(*) begin
3.   if (a || (b && c))
4.     p = <primary_input>
5.   else
6.     p = e;
7. end

Mutant for line#6
Conclusions

• Formal Coverage must fit with Simulation Coverage
  • Same metrics, same meaning

• Formal verification in practice:
  • BMC is the primary workhorse of practical formal verification
  • Checkers are complex Verilog, simple SVA
  • Abstraction Models are key to increasing formal coverage
Thanks

- Adnan Aziz
- Sandesh Borgaonkar
- Richard Boulton
- Choon Chng
- Harry Foster
- Vineet Gupta
- Anton Lopatinsky
- Deepak Pant
- Philippa Slayton
- Shashidhar Thakur

Contact email: vigyan@oskitech.com