Formal Verification at IBM

Applications and Algorithms

Workshop on Making Formal Verification Scalable and Useable

Chennai Mathematical Institute, 9th January 2013
Outline

- Hardware Verification Basics
  - Verification at IBM
  - Formal Verification at IBM
  - Formal Verification: Past, Present, and Future Directions
Hardware Verification: Why so Important?

1) Hardware bugs are **expensive** to fix
   - ~$1M per bug; lost time-to-market; lost marketshare
   - Software bugs are cheap & easy to fix – *look at # Windows patches!*

2) Performance-critical nature of HW entails unique challenges
   - HW is often power-optimized, timing-optimized, pipelined, multi-threaded, …
   - Non-functional design artifacts complicate verification
   - Arithmetic operators (e.g. multipliers) are often bit-optimized
     - Very difficult to prove that bit-optimization preserves functionality!

3) Verification now dominates semiconductor development cost
Verification Complexity: *Intractable!*

- Advances in verification technology + methodology have come a long way to cope with this inherent feasibility
- Nonetheless, *a very active area of research with great industrial needs!*

![Graph showing exponential growth with explanations](image-url)
Introduction to *Hardware Verification*

- Numerous types of **verification** relevant to hardware design

always @ (posedge clk) begin
  if ( r ) then p <= 0
  else p <= p+1;
end if;
end

- Also timing analysis, circuit analysis, protocol analysis, …
Introduction to **Hardware Verification**

- RuleBase SixthSense Edition: a tool for *logical* implementation verification
- 1) *Functional verification*: property checking or model checking
- 2) Combinational and *sequential equivalence checking*

May also be applied to architectural models, protocol models, software-like models, … *As long as they are synthesizable*
Validation and Verification Techniques

Simulation and Acceleration
- Explicit-state guided random walk
  - Scalable to **HUGE** designs
  - **Mature** methodology + tools for high coverage
  - %Coverage inherently **very limited**
    - Misses bugs; never **complete**

Formal Verification (FV)
- **Exhaustive** state coverage via symbolic algos
  - Yields (corner-case) bugs or proofs
  - Capacity-limited to moderately-sized designs

Semi-formal Verification (SFV)
- **Combine** symbolic + explicit search
  - Exposes corner-case bugs on large designs
  - Only yields **bounded** proofs
Outline

- Hardware Verification Basics

- Verification at IBM
  - Formal Verification at IBM
  - Formal Verification: Past, Present, and Future Directions
Verification Technology at IBM

**RTL (VHDL, Verilog)**

- **Language Compile Model Build**
- **Driver/Checker Assertions**
- **PSL etc.**

**Physical VLSI Design Tools / Custom Design**

**Cycle-Based Model**

- **Boolean Equivalence Check (Verity)**
- **(Semi-) Formal Verification (RuleBase SixthSense Edition)**
- **Software Simulator (MESA)**
- **Hardware Accelerator (Awan)**
- **Hardware Emulator**

- **Test Program Generator (GPro, X-Gen)**
- **C++ Testbench**

**Constrained Random Testbench**
Formal Verification at IBM

- Vision: Bring FV to the masses
  - Common infrastructure → Trivial learning curve, resource savings
  - Shared / reusable verification IP → High ROI, tight integration
  - High scalability → Improved productivity

Approaching 200 users per month, >>100k sessions per month
Contrast to early days of model checking at IBM: <10 expert users

Amortize R&D cost → Higher value proposition

- Critical applications by dedicated FV experts still occurs and is very valuable
- Though strong push to enable applications by non-FV experts
Integrated Approach: Design

- Assertion-based Verification (ABV) \textbullet\ Designer-level Verification (DLV)
  - Designers capture assumptions as assertions; used in basic verification
  - High Return on Investment (ROI)
    \begin{itemize}
      \item Facilitates design process: bugs caught \textbf{immediately} vs only after IP integration
        \begin{itemize}
          \item Early design exploration and debug
        \end{itemize}
      \item Easier debug: failed assertion immediately localizes failure or bad assumption
        \begin{itemize}
          \item Valuable even when reused in alternate verification disciplines
        \end{itemize}
      \item Accelerates design closure: some corner-case bugs flushed out early
    \end{itemize}
  - Serve as \textit{documentation}

Assertion-Based Verification

- Enables Integrated Checking
- Enables Stimulus

Designer-Level Verification

- Simulation
- H/W Accel
- Semi-Verification
- Formal Verification

Block-Level Verification

Simple Driver

Complete Driver
Integrated Approach: Verification

- FV plans drawn collaboratively with design and simulation teams
  - LRUs, Arbitration, Debug Buses, … done purely with formal
  - Harder-to-stress design components / functionality targeted heavily with formal
  - Optimize testplans: no need to target FV-covered behavior in sim

- Common verification model facilitates:
  - Leveraging FV to hit hard-to-cover simulation behaviors
  - Establishing FV testbench for bug reproduction (e.g., post-Silicon analysis)

<table>
<thead>
<tr>
<th>PB Unit Testplan</th>
<th>Category</th>
<th>Main Task</th>
<th>Category Detail</th>
<th>Description</th>
<th>Implementation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PB Unit</td>
<td>Arbiter Grants</td>
<td>Performance</td>
<td>Arbiter grants when able</td>
<td>C++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB Macro</td>
<td>Central Cmd</td>
<td>Formal</td>
<td>Identify starvation scenarios for various throttle and I/O rate</td>
<td>VHDL</td>
<td>Investigate complete matrix corresponding to all parameter values.</td>
<td></td>
</tr>
</tbody>
</table>
Hierarchical Verification Progression

VBU = Virtual Bring-Up (chip)
VPO = Virtual Power-On (system)
Outline

- Hardware Verification Basics
- Verification at IBM
  - Formal Verification at IBM
- Formal Verification: Past, Present, and Future Directions
Hierarchical Verification Progression

VBU = Virtual Bring-Up (chip)
VPO = Virtual Power-On (system)

1. Formal Verification
2. Software Simulation
3. Hardware Acceleration
4. Hardware Emulation

- VBU Level
- VPO Level
- System Level
- Chip Level
- Element Level
- Unit Level
- Block Level

Inter-chip interactions
Pervasive verification
Protocol analysis
Recreate bring-up fails
Starvation free arbitration
Defined interfaces
End-to-end check (e.g. FPUs)
“Deep dive” FV
Obtain proofs
Find corner case bugs

FV brings value across hierarchy
Verification Progression (1)

- **Block Level**

  - Targeted “deep dive” driven by knowledge of the micro-architecture
    - Rigorous documentation often lacking at this level
  
  - *Formal / Semi-formal verification leveraged heavily at this level*
    - Designer-level verification
    - FV engineer focus for mission-critical logic / functionality, hard-to-cover in sim
  
  - Small size ⇛ reliable proofs
  - Controllability ⇛ corner cases easily exercised

![Diagram of Verification Process]

\[ \text{Block Level} \rightarrow \text{Driver} \rightarrow \text{Design-Under-Test} \rightarrow \text{Checker} \rightarrow \text{Testbench} \]
Verification Progression (2)

- Functional Units
  - Informal verification: biased random tests directly against unit interface
    - Transaction-, Instruction-based
  - Formal / Semi-Formal verification applied selectively at this level
    - Similar to block-level, though larger size → capacity challenges
    - Better-documented / simpler interfaces, reusable drivers / checkers
    - Reference model-based end-to-end check

  - Fixed- / Floating-point Unit, Memory Controller, …
Verification Progression (3 & 4)

- **Element and Chip Level**
  
  - Informal verification: transactions, pre-generated test programs
  
  - (Semi-) formal verif used to verify multi-unit / core interactions, architectural aspects…
    - Reuse RTL models with suitably abstracting blocks / units with behaviorals
    - Multi-unit models with *heavy black-boxing* / over-constraining
    - Verify unit interconnections, clock domain crossings, …

  - *Hangs, stalls, bus protocols, arbitration*…
Verification Progression (5)

- **System Level**
  - Informal verification: Pre-generated test-programs
    - Multiprocessor models / tests
    - I/O chips interactions, asynchronous aspects
  - Formal methods applied to study chip interactions
    - High-level analysis of protocol models
    - Traffic flow, asynchronous interfaces, timing protection windows, deadlocks…
Quality Refinement Process

Because *controllability*, *state coverage* is higher, and *cost* of a bug is lower, at lower levels:

- Every major bug find at higher level is treated as *escape* of lower level

- Lower level team gets feedback to reproduce problems
  - Harden lower level environments
  - Reproduce with targeted block-level checkers
    - Prove fixes with formal verification
Sequential Equivalence Checking

Sequential Equivalence Checking (SEC)

- OLD Design
- NEW Design
- Initialization Data
- Input Constraints

Sequential Equivalence Checker

Supports arbitrary changes that preserve IO behavior

Retiming, power optimization, logic minimization, ...

Hierarchical application enables high scalability

Game changing application of FV

End-to-end verification of entire chips

Invaluable productivity advantage, resource savings

Unbounded proofs are critical in SEC!
Model checking with RuleBase SixthSense Edition

Environment, Driver

Assertions, Properties

RuleBase
SixthSense

DUV

Fail
+ Countercexample

Pass
+ Witness

Pass vacuously

Bounded pass

[1:n]
Supported languages

- Hardware description languages
  - Verilog, VHDL, Mixed
  - GDL (mainly for protocol verification)

- Assertion languages
  - PSL: standalone checker or embedded in HDL
  - SVA

- Verification directives
  - Assertions, Coverage points
  - Assumptions
  - Full liveness, fairness, restrict support

- Support for various trace browsers

- High capacity via **Transformation-Based Verification**
Transformation-Based Verification

- Encapsulates engines against a *modular API*
  - Transformation engines, proof engines, falsification engines

- Modular API enables *maximal synergy* between engines
  - Each (sub)problem may be addressed with an arbitrary sequence of algos
  - Motivation: every problem is different; different algorithm sequences may be exponentially more / less effective on a given problem

- **Incrementally chop complex problems into simpler problems, until tractable for core verification algos**
Example Transformations

- **Retiming**
  - Forward
  - Backward

- **Localization**

- **Redundancy removal**

- **Logic Rewriting**
Transformation-Based Verification

Problem decomposition via *synergistic transformations*

- **Design + Driver + Checker**: 140627 registers
- **Combinational Optimization Engine**: 119147 regs
- **Phase Abstraction Engine**: 79302 regs
- **Localization Engine**: 1320 regs
- **Retiming Engine**: 189 regs

Parallel algo exploration, (sub)problem solution

Counterexample consistent with original design

Optimized, phase abstracted, localized trace

Optimized, phase abstracted trace

Counterexample consistent with original design
Transformation-Based Verification

Problem decomposition via synergistic transformations

Transformations are completely transparent to the user – internally enable exponential speedups!

All verification results are in terms of original design

Parallel algo exploration, (sub)problem solution
Example Engines

- Combinational rewriting
- Sequential redundancy removal
- Min-area retiming
- Sequential rewriting
- Input reparameterization
- Localization
- Target enlargement
- State-transition folding
- Circuit quantification
- Temporal shifting + decomposition
- Isomorphic property decomposition
- Unfolding
- Speculative reduction
- Symbolic sim: SAT+BDDs
- Semi-formal search
- Random simulation
- Bit-parallel simulation
- Symbolic reachability
- Property-directed reachability
- Induction
- Interpolation
- Invariant generation
- Array abstraction

- Expert System Engine orchestrates parallel optimal engine selection
- If there is a useful verification algorithm, RuleBase SixthSense Edition likely has it!
  - *Much innovation*: necessity is the mother of invention; IBM has deep verification needs!
  - Also *much collaboration*!
Outline

- Hardware Verification Basics
- Verification at IBM
- Formal Verification at IBM
  - Formal Verification: Past, Present, and Future Directions
Model Checking Capacity vs Time

Design size at which some useful results could be expected from FV tool

Caveat: not guaranteed capacity; 1) some tiny problems are unsolvable! 2) includes bounded proofs

Very incomplete list; cumulative capacity trend leverages earlier innovations + SW engineering
Formal Verification Evolution @ IBM

Early Times
- Applied to small logics (~100s of registers)
- Manual Intensive w/ dedicated resources
- Required setting up of complex drivers

Middle Ages
- Advent of SFV, Parallel, SEC
- Larger logics verified; higher coverage
- Same “look and feel” as simulation
- SEC key to many newer methodologies

Modern Era
- Large scale FV application
  - Integrated approach / DLV
  - Out-of-the-box methodologies
  - High speed, capacity toolsets

The Future…
- Avoid duplicate work
  - Reusable methodologies / IP
- Automation, automation…
- Stay tuned!

SFV: Semi-formal verification
SEC: Sequential Equivalence Checking
DLV: Designer-level Verification

High tool capacity has enabled profound methodology impact
What is RuleBase: SixthSense Edition?

- **RuleBase PE** (IBM Research)
  - Robust algorithms for property checking
  - Advanced usability features, rich language support

- **SixthSense** (IBM EDA)
  - Focus on **very high capacity** via *Transformation-Based Verification*
  - Robust support for property checking + sequential equivalence checking

- **RuleBase: SixthSense Edition**
  - Usability, rich language support, diverse application domains
  - **Order of magnitude improvement in capacity**
    - Cited as the strongest model checker + sequential equivalence checker in the world
  - World-class R&D team with a unified focus on continued capacity boosts
RuleBase SixthSense Edition: R&D Team

- World-wide R&D team

- Active development since 1993

- Unique patented transformation-based verification architecture

- >100 granted patents

- >50 technical conference papers

- **Numerous key verification innovations developed through this project**
  - Equivalence checking, on-the-fly checking, And / Inverter Graphs, time-sliced simplification vs SAT solving, Transformation Based Verification, cone-of-influence reduction, speculative reduction, ...

  - Incubator of PSL, IEEE 1850
RuleBase: SixthSense Edition  R&D Team

External collaborations

California
Colorado
Minnesota
New York
Austin, Texas
Ireland
Italy
Sweden
Austria
India
Israel
FV @ IBM: Impact Highlights

- This technology has become *essential* to IBM’s business
  - Successful FV deployment *mandates* high capacity; drives R&D
  - Tight synergy between formal and informal verification teams + design teams
    - Spec reuse across teams
    - Push for spec + methodology reuse across projects
  - Continually finding new application domains where FV *displaces* simulation

- Approaching 200 FV users / month; >>100k sessions / month
FV @ IBM: Impact Highlights

- SEC has become a huge “commercial win”
  - Technology remaps, “IP import” projects completely forgo functional verification
  - *Timing bringdown* design phase greatly simplified by SEC capability
  - Enables late / aggressive changes that otherwise would not be tolerated

- Designer-level applications are a huge “intangible win”
  - Critical to *shift bug-count left / first-time correct silicon* mantra
    - Accelerates successful higher-level verification bringup
  - Also used for exploration of design optimization opportunities
FV @ IBM: Impact Highlights

- Verification teams migrating from small blocks to larger blocks / units
  - *Scalability* is enabling FV to verify *functionality* vs verify small-enough *blocks*
  - Verify against more stable and meaningful design interfaces
    - Less testbench bringup effort
    - Fewer *testbench* bugs vs *design* bugs

- Silicon failures almost 100% addressed with FV (if *logical* failures)
  - An additional driving force to establish early formal testbenches
Open Problems: *Call for Research Focus*

- **Many open problems in design + verification**
  - *HW verification is not a solved problem*
  - Many unsolvable problems; manually-intensive to cope with these

- **Old open problems:**
  - Improve bit-level verification, falsification algorithms !
  - Improve bit-level synthesis algorithms !
  - Improve equivalence checking techniques !

- Bit-level techniques remain primary workhorse in industrial HW verification
- Verification of bit-optimized arithmetic circuits is particularly troublesome
  - The remaining achilles-heel of CEC
Open Problems: *Call for Research Focus*

- **Newer open problems**
  - Improve higher-level verification algorithms (e.g. SMT) !!
  - Improve higher-level synthesis techniques !!
    - Optimize integrated theory solvers due to heterogenous nature of HW
  - Improve higher-level equivalence checking techniques !!

- Goal – enable higher-level design without manually-derived ref model
- Grand challenge: application of higher-level algos to bit-level designs

- “Someday Moore’s Law will work for, not against, the verification community” Allen Emerson
- Requires substantial innovation! *Help us achieve this goal !!!!*
References

- Project homepage

- Technical publications
  - [https://www.research.ibm.com/haifa/projects/verification/SixthSense](https://www.research.ibm.com/haifa/projects/verification/SixthSense)

- Contact:
  - Jason Baumgartner: baumgarj@us.ibm.com
  - Viresh Paruthi: vparuthi@us.ibm.com
  - Ambar Gadkari: ambar.gadkari@in.ibm.com
  - Pradeep Nalla: pranalla@in.ibm.com
  - Sivan Rabinovich: sivanr@il.ibm.com