Industrial SFV Experience – Last Year’s Lecture

- After the initial brouhaha, not much is heard of SFV applications nowadays
  - Tools tend to focus more on improving BMC methodology
    - Increasing search depth capability by 1 will expose more bugs
  - Commercial tools no longer advertise the semi-formal capability?
  - Cynical view: merely prolongs the agony of sim-based methodology
- SFV is still a very useful technology
  - Extends the bug hunting capability of BMC to deep bugs in large designs
  - Enabling technology for widespread use of formal techniques
- Needs methodology and tool support to be effective
Verification sign-off using Formal Verification

- FV has become mainstream verification methodology
  - Deployed in all major companies that develop complex SoCs
- FV has many advantages over dynamic simulation
  - Ability to generate exhaustive proofs
  - Simpler testbench structure
  - Faster time to setup and faster time to find counterexamples
- Is there any advantage of doing block-level verification sign-off using FV?

Acknowledgment: Hao Chen et.al, DVCON 2019
Verification sign-off using Formal Verification

To-enable signoff, we need a signoff metric

- **Coverage** is the obvious answer
  - Has the advantage of being the common metric with dynamic simulation

- How about a coverage-driven FV methodology?
- FV is supposed to cover 100% state-space, why need coverage?
  - Unintentional over-constrains in FV environment
  - Design-complexity – cannot achieve exhaustive coverage leading to bounded proofs

- What algorithms can FV tool use to assert coverage goals?

---

Acknowledgment: Jasper Gold User Group 2018
Verification sign-off using Formal Verification

Verification Sign-off using Formal Verification is IMPOSSIBLE without using semi-formal verification

- Let’s learn SFV

Outline

- Semi-formal verification (SFV) – What, Why, How?
- Brief overview of various semi-formal techniques
- Industrial SFV Experience
- Verification Tools with semi-formal technologies
Semi-formal Verification (SFV) – What, Why, How

- What is semi-formal verification (SFV)?
  - An attempt to combine the completeness of formal techniques with the speed, capacity and ease-of-use of simulation
  - Leverage formal techniques in a resource-bounded way

- Why are semi-formal techniques needed?
  - Critical for verification sign-off using FV
  - Corner-case bugs too complex for sim and too deep for formal
  - Critical for deep bugs
  - Key to scaling formal algorithms to large, complex designs

- How does semi-formal techniques work?
  - Augmenting simulation using formal techniques
  - Guiding simulation using formal techniques

SFV by Augmenting simulation

Verification Problem: Check for queue overflow

- FV techniques such as bounded model checking (BMC) cannot go deep enough
- SIM has to get lucky – cannot consider all possible scenarios
- BMC can do exh. search from a state such as queue is 1/4 full
- Can Sim get the design to queue is 1/4 full state? **YES!**

Acknowledgment: J. Baumgartner, IBM
SFV by Augmenting Simulation

Methodologically: Manual definition of guideposts


Automatic or Tool Driven

- IBM SFV tool RuleBase-SixthSense
- Jasper-Gold (Cycle-Swarm technique)
- Synopsys FV toolkit
- Mentor Graphics FV toolkit

SFV by Augmenting Simulation: Challenges

- Formal search is only effective if it is triggered near a fail
  - Cannot improve falsification capability otherwise
  - Techniques that make simulation “smarter” are applicable
    - e.g., better input pattern generation using biases

- Approaches
  - **State prioritization**: try to trigger iterations from new/interesting states
    - Apply rarity-analysis to find interesting states
  - **Light-houses/Stepping-stones**: automated generation of guideposts towards fail
    - Unhit design states can be used as guideposts
    - Can use formal analysis to assert the lighthouses
    - Yalagandula et. al., “Automatic lighthouse generation for directed state space search”, DATE 2000
  - **State-swarming technology in Jasper-Gold is essentially state-prioritization**
Rarity-guided Simulation (Nalla et.al, ICCAD 2016)

- Synergistic interaction of guided bit-parallel simulation and SAT-based bounded model checking (BMC)
- Each simulation word = 64 scalar bit-level simulations, evaluated in parallel (exploits data parallelism)
  - Often achieve highest throughput with 50-1000 words
- Many heuristics to select "interesting" states for later search iterations, to guide simulation + BMC

- BMC has the power to cover an astronomically larger number of states than sim
- Seeding BMC into the right states yields deep bugs impractical using BMC alone
- Coupled with "lighthouses" enable BMC to guide sim into very improbable states

- Guiding sim between iterations is important to direct it into new vs redundant search
- Guidance is even more important in testbenches with "constraints" or "assumptions"
  - Sim will diverge into "dead-end states" which do not satisfy the constraints otherwise
Rarity-guided Simulation (Nalla et.al, ICCAD 2016)

- Rarity is a measure of how often a state occurs during random simulation
- Precise state recording is unscalable: approximate using register-partition states
- Idea: seed search iterations into diverse/rare states, for higher collective coverage

```
//Weight calculation w.r.t. partition state coverage counters
for each simulated state {
    for each register subgroup { cnt = getCounterValue(state, subgroup); stateWeight += 1 / (cnt*); }
}
```

References: 1) “Rarity-based guided state space search” – Ganai et al; 2) “Using speculation for Sequential Equivalence checking” – Brayton et al

Rarity-guided Simulation (Nalla et.al, ICCAD 2016)

- Heuristic 1: Select Entire “Rarest Timestep”
  - Timestep-Based Rarity = Sum of rarity weights of each bit-parallel state in that timestep
  - Seed entire “rare bit-parallel vector” into next sim iteration
  - Like “backtrack to earlier timestep” and restart sim with new random inputs
  - BMC seeding: select rarest state in that rare timestep
Rarity-guided Simulation (Nalla et.al, ICCAD 2016)

- Heuristic 2: Select Rarest States within “Rarest Timestep”
  - Select rarest timestep; sort states therein w.r.t. rarity
  - Selects top \(n\) rare states therein for next round of bit-parallel simulation
    - Recall: random input generation → higher coverage even when reusing same starting state
    - Bit-parallel sim evaluates them efficiently in parallel
  - BMC seeding: select the rarest state therein

- Selection of best rare states across all timesteps
  - Particularly useful in constraint-based testbench: few vectors may be valid in a timestep
  - Improves BMC coverage, since doesn’t miss single-rarest state when selecting rarest timestep
**SFV by Augmenting Simulation: Other Approaches**

- **Target Enlargement**: compute a few preimages of the target states, in order to create a larger set of target states
  - Improves the probability of hitting the target states
  - Simulation can use greedy search strategies

**Diagram:**

- Initial state (init)
- States B0, B1, B2, B3, B4
- Target state

**SFV by Guiding Simulation**

1. Create an abstract design
2. Perform exhaustive formal search
   - Partition the reachable state set into onion rings
   - $i$-th onion ring can reach the target in $i$ steps
3. Guide simulation to move the (concrete) simulation state to one that maps into the next closer onion ring
   - All concrete paths have corresponding abstract paths, but not vice-versa

**Diagram:**

- Abstract pre-images
- Concrete simulation trace
- Target state
SFV by Guiding Simulation: Challenges

1. The abstract design is too over-approximate
   - No legitimate concrete trace exists that maps to abstract trace
2. A majority of concrete states hit are dead-end states
   - There is no path from dead-end state to a concrete state that maps to the next abstract onion ring
3. A short abstract trace may correspond to very lengthy concrete trace
   - The abstraction is too coarse (over-approximate)

SFV by Guiding Simulation: Overcoming Challenges – Abstraction

- Abstraction is the key to overcoming the challenges
  - Retain enough behavior to efficiently guide simulation
  - Needs to be small enough to enable exhaustive search
- Use automatic abstraction techniques such as localization
  - CEGAR (counter-example guided abstraction refinement) can be used to automatically refine the abstraction
  - Applicable across wide-variety of testbenches
  - Abstract models generated through localization tend to get large quickly
  - Exhaustive search quickly hits a brick wall
SFV by Guiding Simulation: Overcoming Challenges – Abstraction

- Abstract using domain knowledge
  - Verification engineer manually abstracts the design
  - Restricts the applicability of the technique
  - S. Shyam and V. Bertacco. “Distance-guided hybrid verification with GUIDO.” DATE, 2006
  - F. M. de Paula and A. J. Hu. “EverLost: A flexible platform for industrial-strength abstraction-guided simulation” CAV’06
- Abstract using data mining and domain knowledge
  - Aims to avoid the pitfalls of manual approach and localization
  - Applicability not demonstrated on industrial testbenches

- Abstract away the data-path to retain control-path registers
  - Most designs have data paths and controllers
  - Most bugs result of infrequent interactions between controllers
  - Simulation attempts to explore as much of the control state space as possible, thus increasing the likelihood of finding bugs
  - Requires high-level design information
SFV by Guiding Simulation: Overcoming Challenges – Dead-End States

- Dead-end states cannot reach the target
- There are no concrete transitions that correspond to the transition in the abstract model
- Simulator cannot detect the dead-end state
  - Should we backtrack or should we try harder?

SFV by Guiding Simulation: Overcoming Challenges – Dead-Ends and Coarse Abstractions

- Dead-ends are exacerbated by the greedy simulation guidance
  - We always choose state from the closest onion ring
- One Solution: Remember multiple states; balance between greed and relaxation
  - Flip a coin to decide on which onion ring to continue the search
- Coarse Abstraction results in lengthy concrete traces
- Simulation ineffective in tunneling between abstract onion rings
  - Use Formal Analysis such as BMC to tunnel through
  - S. Shyam and V. Bertacco. “Distance-guided hybrid verification with GUIDO.” DATE, 2006
Industrial SFV: Methodology and Tool Support

- Synergistic Simulation/SFV Methodology needed
  - Common Design Model for Sim and SFV
    - There should not be any semantic gaps
  - Common Design Partitions / Units
    - FV typically applied to macro (sub-unit) level that require specifications at non-documented, fluid interfaces
  - Common Designer Assertion/Coverage Specifications
    - This has largely been achieved on the assertion side, how about coverage?
  - Common environment specification / Testbench drivers
    - Implies synthesizable testbenches?
- A tool that can scale to design partitions that have documented specifications

Industrial SFV: Coverage

- What are the questions you want answered using coverage metrics?
  1. Does my FV environment allow all possible legal stimulus?
  2. Is my set of Assertions complete? Do they cover all possible design behavior?
  3. How much of the design space is covered by my proven assertions?
  4. Are the bounds obtained on undetermined assertions sufficient?
Industrial SFV: What are the Coverage events?

- Coverage events fall into two main buckets
  - Code Coverage
    - Branch coverage
    - Statement coverage
    - Expression coverage
    - Toggle coverage
  - Functional coverage
    - Property (SVA/PSL)
    - Covergroup

Industrial SFV: Stimulus coverage

- Attempt to hit all cover events
- Try to prove events that are unreachable
- Ideally all cover events must be hittable
- Unreachable cover events can be either due to:
  - Dead code: impossible for any stimulus to hit
  - Unreachable due to overconstraining environment
- Which engine is critical for ensuring stimulus coverage closure?
  - *Semi-Formal Engine of course!*
Industrial SFV: COI coverage

Acknowledgment: Jasper Gold User Group 2018

Industrial SFV: Proof coverage

- SAT-based engines can generate a proof-core; logic responsible for proving correctness of a property

Acknowledgment: Jasper Gold User Group 2018
Industrial SFV: Are my bounds sufficient?

- “N” cycle bounded proof implies that all states reachable within “N” cycles have been analyzed
- Determining whether the bounds are sufficient is more Art than Science
- The current best approach is specifying functional cover events that force deep exploration of state space
  - Use SFV engines to hit the cover events
- Using a combination of designer knowledge and data from previous verification efforts, one can possibly predict a good enough “N”
  - Possible application of ML/DL?

RuleBase-SixthSense: IBM’s SFV toolset

https://www.research.ibm.com/haifa/projects/verification/SixthSense/

**RuleBase-SixthSense is a system of cooperating algorithms**

- Semi-Formal engines
- Formal engines
- Transformation engines: simplification / abstraction algorithms

**Transformation-Based Verification (TBV) framework**

- Exploits maximal synergy between various algorithms
- Redundancy removal, retiming, induction, localization, ...
- Incrementally chop problem into simpler sub-problems until solvable
The Case for Transformation-based Verification

- FV – exhaustive, but needs exponential resources w.r.t size
- High performance design particularly difficult to verify
  - Speed, area and power concerns demand subtle optimization
  - Complex control, pipelining logic increases verification complexity
- Key Insight: Use automatic transformations to extract the simple underlying model
- RuleBase-SixthSense framework: Synergistically leverage various transformations to simplify and decompose complex problems

Example Transformation flow

140627 registers

Dr. Hari Mony, Real Intent, March 28, 2019
Example RuleBase-SixthSense Engines

- Boolean Reduction
- Sequential redundancy removal
- Min-area retiming
- Sequential rewriting
- Input reparameterization
- Localization
- Target enlargement
- State-transition folding
- Isomorphic property decomposition
- Fast Forward Engine

- Unfolding
- Liveness-to-Safety Transform
- Semi-formal search
- Symbolic simulation: SAT+BDDs
- Symbolic reachability
- Induction
- Interpolation
- Property Directed Invariant Generation (IC3)

Expert System Engine automates optimal engine sequence experimentation

Boolean Reduction Engine (BRN)

- Combinational redundancy removal techniques: BDD- and SAT-sweeping
- Can be enhanced through leveraging observability don’t cares
**Boolean Reduction Engine (BRN)**

- Logic rewriting algorithms, to simplify logic expressions
- Lowering gate count greatly enhances SAT-based reasoning
- Also tends to enhance reduction potential of other algorithms
- A. Mishchenko et. al., “DAG-aware AIG rewriting a fresh look at combinational logic synthesis”, DAC 2006

---

**Boolean Reduction Engine (BRN)**

- Ternary Simulation: simulate an AIG over 3-valued logic
  - Sequence of 3-valued states
  - Converge when the current state is a subset of past states
  - Over-approximate reachable state set can be used for identifying constants/equivalent signals
Sequential Redundancy Removal (EQV)

- Sequential Redundancy Removal Algorithm: A Recap
  1. Guess a set of redundancy candidates; sets of gates that are expected to be functionally equivalent (modulo inversion)
  2. Attempt to prove redundancy candidates accurate
  3. If any redundancy candidate cannot be proven, partition the groups to separate those that cannot be proven equivalent; goto Step 2
  4. The current groups reflect true redundancy; simplify the netlist

- Superset of BRN reductions, more expensive than BRN
- Very effective for bug-finding as well as proofs
- Mony et. al., “Exploiting Suspected Redundancy without Proving It”, DAC 2005
- Mony et. al., “Speculative-Reduction based Scalable Redundancy Identification”, DATE 2009

State Transition Folding (MOD)

- MOD: a structural state folding engine
  - Each state transition in the transformed design corresponds to multiple original transitions
  - Used to perform clock abstraction
    - Replaces Master-Slave with a flip-flop
  - Very powerful in designs with multiple clock domains
  - Reduces the minimum depth at which targets can be hit
- Per Bjesse, James H. Kukula: “Automatic generalized phase abstraction for formal verification”, ICCAD 2005
Transformation (RETiming)

- RET: Min-area retiming: Reduces the number of registers by moving them across combinational gates
  - Very powerful for deeply-pipelined as well as feed forward designs
  - May increase AND and Input count, use in association with BRN, CUT

Liveness to Safety (LIV)

- LIV: Liveness to Safety transformation
  - Properties categorized as either Safety or Liveness
  - Safety: Is my division result correct?
    - Finite counterexample
  - Liveness: Will the request eventually get a grant?
    - Infinite counterexample: to illustrate that grant can never occur
  - Traditional liveness checking very expensive
  - Can transform the netlist to convert liveness to a safety property
    - But doubles the number of state elements in the design
  - Armin Biere, Cyrille Artho, Viktor Schuppan: “Liveness Checking as Safety Checking”
### Transformation (Localization)

- AXE: localization; remove logic not relevant to the target
  - Overapproximate – proofs in localized netlist valid
  - Necessary to complete proofs on huge designs
- Core Idea: Use SATisfiability-based analysis to identify logic needed to prove target unreachable for specific number of cycles (say N)

![Diagram of transformation (localization)](image)

---

### Native Memory Modeling

- Native memory array support
  - No more bit-blasting!
  - Much faster semi-formal bug hunting
  - Proof complexity = #reads relevant to property
    - Not #addresses in array
  - Enables reasoning about software-like systems
  - Enables various advanced abstraction techniques
Expert System (XPT)

- Discovering well-tuned transformation flow non-trivial
  - Need to understand engines/options, rate their effectiveness
  - Intelligent experimentation necessary to find conclusive flow
- Don’t want to delve into the engines ⇒ use the XPT engine
- XPT automates the experimentation needed to solve tough problems
- Every problem is different, can the XPT engine itself be tuned?
  - YES!

Why Parallel XPT

- Discovering a well-tuned engine flow is nontrivial
- Every problem different; complex problem requires 100’s of transforms
- XPT automates this experimentation by deploying rules
  - Deployed rules setup for commonly encountered problems
  - Deployed rules not aggressive
  - Certain strategies only work for a small subset of problems
  - XPT tries multiple strategies – but in prioritized manner
- Solution: Parallel XPT with multiple strategies in coordinated manner
Parallel Orchestration

SFV Algorithms in RuleBase-SixthSense

Augmenting Simulation
- Ability to manually define guideposts and have the tool step through them
- Target Enlargement
- Interleaved BMC and random simulation
  - State prioritization
  - Automatic generation of lighthouses

Guiding Simulation
- Automatic generation and refinement of abstract models using localization
- Ability to tunnel between onion rings using BMC
- Heuristic guidance strategy to avoid deadend states
Applications of SFV

Virtually all RuleBase-SixthSense applications benefit from semi-formal search

**Assertion-based verification**
- Typically done by designers with lesser experience level with FV and toolset
- Testbenches developed with little thought about “proof strategy”
- SFV very useful to wring out bugs

**Reference-model based verification**
- Comprehensive checks, usually implemented as an abstract reference model
- For larger units, often benefits from SFV to wring out early bugs

Applications, Cont’d

**Silicon-failure recreation efforts: When a chip misbehaves …**
- On-chip debug facilities offer partial insight into cause
- Usually have a good idea of property to check, “buggy region”
- SFV very useful since often requires a fairly large design slice, and bug-hunting vs. proving is “the mission”

**Coverage analysis**
- Leverage formal algorithms to help simulation reach hard-to-hit scenarios

**Sequential equivalence checking**
- Semi-formal search useful to find mismatches, assist in guessing equivalent gates
SFV: Conclusions and Future

- SFV is an enabling technology for wide-spread FV usage
- FV-based Verification sign-off is impossible without SFV

Future Research

- The future of SFV is bright!
- Take advantage of new machine architectures for improving simulation throughput (bit-parallel simulation)
- Enhance simulation through intelligent pattern generation
- Methodology/Algorithms to determine if bounded coverage is sufficient