Formal Verification: As I Know it

Sudhakar Surendran
Technical Lead
MCU, Texas Instruments
Agenda

• Acknowledgements
• What I know as Formal Verification (FV)
• Why & how I started on FV
• How I do FV
• Where I found FV to be effective
• My learning's in using FV
• Where I see FV heading
• Conclusion
Acknowledgements

- Dr. Subir K Roy
- Dr. Raj S. Mitra
- Arif Mohammed
- Harish Maruthiyodan
- Lokesh Babu Pundreeka
What I Know as Formal Verification (FV)

- Formal analysis that proves/disproves Design meets the Formal Specification
  - Design is the DUT (Design Under Test) and is usually the RTL implementation
  - Formal Specification captures in a precise and unambiguous manner the expected design behavior
    - Usually captured in terms of properties in a proper language like SV/PSL/OVL. Also known as Assertion Based Verification (ABV)
    - ABV is not the only way to do FV
  - Formal analysis is performed using a combination of algorithms:
    - ATPG (Automated Test Pattern Generator)
    - BDD (Binary Decision Diagrams)
    - SAT (Satisfiability)
Why & How I Started on FV

• Why I started on FV? (Re-visited in Learning's)
  – Complete design coverage for a given property
  – No additional TB to be developed
  – Faster verification closure
  – RTL designer can code the design intent reducing DV effort

• How I started on FV?
  – Training : Online resources and expert training
  – Toy examples
  – Started with simulation based ABV and moved on to Formal ABV
  – Full stand alone IP-DV
How I do FV

• Integrated Formal and Simulation Design + Verification Planning
  – Planning very critical for FV to extract full efficiency and quality benefits

• Analyze, partition and parameterize the design for FV
  – Analyze the uArch with designer
    • Identify weak or critical or combinatorial portions of the design
    • Control sections are better matched for FV. (a) Usually buggy and (b) Easy for FV algos
    • Data paths can be included in FV provided design is properly is partitioned and parameterized. Helps to contain the flop count with limits of the tool
    • Serial IPs like I2C may not be best candidate but can still do FV with additional DV abstraction/modeling
  – Partition the design
    • Enables use of “Assume-Guarantee” (AG) approach
      – Breaks the path into smaller manageable chunks and write property to verify these chunks
How I do FV

• Integrated Formal and Simulation Design + Verification Planning
  – Planning very critical for FV to extract full efficiency and quality benefits

• Analyze, partition and parameterize the design for FV
  – Analyze the uArch with designer
    • Identify weak or critical or combinatorial portions of the design
    • Control sections are better matched for FV. (a) Usually buggy and (b) Easy for FV algos
    • Data paths can be included in FV provided design is properly is partitioned and parameterized. Helps to contain the flop count with limits of the tool
    • Serial IPs like I2C may not be best candidate but can still do FV with additional DV abstraction/modeling
  – Partition the design
    • Enables use of “Assume-Guarantee” (AG) approach
      – Breaks the path into smaller manageable chunks and write property to verify these chunks

Assume   Guarantee (Assertion)
How I do FV

• Integrated Formal and Simulation Design + Verification Planning
  – Planning very critical for FV to extract full efficiency and quality benefits

• Analyze, partition and parameterize the design for FV
  – Analyze the uArch with designer
    • Identify weak or critical or combinatorial portions of the design
    • Control sections are better matched for FV. (a) Usually buggy and (b) Easy for FV algos
    • Data paths can be included in FV provided design is properly is partitioned and parameterized. Helps to contain the flop count with limits of the tool
    • Serial IPs like I2C may not be best candidate but can still do FV with additional DV abstraction/modeling
  – Partition the design
    • Enables use of “Assume-Guarantee” (AG) approach
      – Breaks the path into smaller manageable chunks and write property to verify these chunks
How I do FV

- Integrated Formal and Simulation Design + Verification Planning
  - Planning very critical for FV to extract full efficiency and quality benefits

- Analyze, partition and parameterize the design for FV
  - Analyze the uArch with designer
    - Identify weak or critical or combinatorial portions of the design
    - Control sections are better matched for FV. (a) Usually buggy and (b) Easy for FV algos
    - Data paths can be included in FV provided design is properly is partitioned and parameterized. Helps to contain the flop count with limits of the tool
    - Serial IPs like I2C may not be best candidate but can still do FV with additional DV abstraction/modeling
  - Partition the design
    - Enables use of “Assume-Guarantee” (AG) approach
      - Breaks the path into smaller manageable chunks and write property to verify these chunks
How I do FV (cont.)

• Analyze, partition and parameterize the design (cont.)
  – Parameterize the design
    • Helps to write scalable properties.
    • Helps to reduce the data path complexity. Useful when design complexity cannot be handled by the tool

• Assertion planning
  – Work with spec owner to write machine readable spec
    • Helps to auto-generate assertions from a template
  – Check Assertion VIPs availability with the vendors e.g. AMBA assertions VIPs
  – Look for opportunities to use FV verification apps from vendors to cover your planned assertions e.g. register check package
  – Break complex design properties into simpler assertions
    • Broken at design boundaries and use AG approach
    • Broken into multiple properties that check mutually exclusive cases of the property
    • Split the check between TB model and Assertion
  – Constraints/Assumptions planned and reviewed with Designer+Spec owner
  – Parameterize assertions similar to design parameterization
How I do FV (cont.)

- Execution planning
  - TB models to enable assertion checks
    - To enable writing simpler assertions
  - Tool and its options
    - Selecting right engine for the design type (can be got from reference guide)
  - Simulation to FV back and forth transitions
    - To check assertion correctness
    - To manage state space explosion
    - Improve coverage at IP level if FV is done at sub-IP level
  - Plan B for cases the FV tool give up
    - Use of abstractions for sections of design that have huge state space
    - Hybrid Simulation+FV approach
  - Coverage tracking and closure
    - Understating how the tool measure the FV coverage
    - Additional properties may have to be written to ensure coverage is indeed met
Where I found FV to be effective

- Combinational logic with minimal sequential elements
  - E.g. Pad multiplexing logic, Module connectivity, Mux and control of Int/DMA selection
  - Can be auto generated from machine readable spec using a assertion template

- Bus-matrix, protocol convertors (bridges)
  - Assertions for bus-matrix can be auto-generated if parameterized VIPs are coded for all the variants of bus protocols
  - Bridge FV verification is best if protocol VIPs exist
    - Difficult to identify all coverage items in simulation

- System critical property checks
  - E.g. RDY signal is never stuck, grant eventually returned, performance critical checks
  - Very critical to ensure SoC does not hang. Worth the extra time spent on FV to get completeness

- Critical state-machines that can be specified with simple assertions
  - E.g. Power state-machines.
  - Complex when looked at the entire system but when partitioned turns to be simple assertions (Smart design partitioning)
Where I found FV to be effective (cont.)

- High risk state-machines and relevant data-path
- RTL optimizations for performance improvements
  - E.g. Pipelining to reduce clock cycle
  - Use non-optimized RTL with model tweaks and abstractions to Equivalence check optimized vs non-optimized RTLs
  - Requires advanced FV knowledge. Used where SLEC is not applicable
- Other non-conventional tasks
  - Coverage exclusion files for simulation coverage closure
  - Un-initialized flop list generation (used in GLS sims)
  - Pattern generation for ATPG closure
  - Fail pattern generation for debug
My Learning's in Using FV

- Planning, planning, planning…

- FV since is not purely a DV activity. Need support from RTL and Spec owners
  - More that what is usually done for Simulation based DV

- Need well thought out plan on assertion coding and correctness checking
  - Easy to miss bugs if not careful compared to Simulation based DV
  - E.g. Using output ports as assumptions without having relevant assertions checks for the same
  - Simple and well documented assertion coding helps in good review feedback and minimize assertion bugs

- Compared to simulation don’t get a feel for how the assertion is performing
  - For novice good to start with Simulation ABV and then move to FV ABV

- Plan to go back to simulation for state explosion cases
  - Or use hybrid approach
My Learning's in Using FV (cont.)

• Why I started on FV? (Re-visited)
  – Complete design coverage for a given property
    • Coverage as good as quality and completeness of properties identified for the design
  – No additional TB to be developed
    • TB models to simplify assertion coding
    • TB Models to abstract environment behavior
    • FV models not as complex as simulation models and hence don’t take huge effort to develop
  – Faster verification closure
    • Complex design may need to spend equal amount of time as Simulation DV
    • Quality will be better than simulation env. Time spent justified for critical modules
    • True for smaller design
  – RTL designer can code the design intent reducing DV effort
    • Skill and mentality change needed to get this
Where I see FV heading

- FV being made a part of IP and SoC verification
- Verification apps and VIPs from vendors and 3rd parties enabling easier FV adoption
- Tools to check assertion correctness or code correct by construction assertions
- FV based tools to verify incremental design change
- With improvement in skill level engg. using in-research products from Universities and consortiums
  - More non-verification uses
  - Analog equivalence checks
  - SW-HW co-verification
Conclusion

• Shared my answers to why, how, where of FV

• My learning on what can go wrong with FV usage and being aware of the same

• My views on where I think FV is heading

• FV is a unique tool when used effectively it can help the DV engg. to improve quality and productivity