Formal Verification

Rony Gutierrez – Mellanox Technologies

Elchanan Rappaport – Gila Logic LTD.

Formal: Property vs. Equivalence

- Equivalence statically compares two designs, often at different levels (e.g. RTL vs. gate-level) to guarantee equivalence.
- Typical uses of equivalence:
  - Confirm no bugs introduced by synthesis
  - Confirm manual changes in gate level haven't introduced errors.
- Equivalence does NOT check for design bugs.
- Property Checking DOES check for design bugs.

Formal - Property Checking

- Providing a formal proof on a mathematical model of the system.
- Exhaustively verify that a set of assertions holds on a design or compute counter examples automatically. All design is abstracted as a big FSM.
  - Tool will search all state space.

Bug Hunting vs. Full Proof

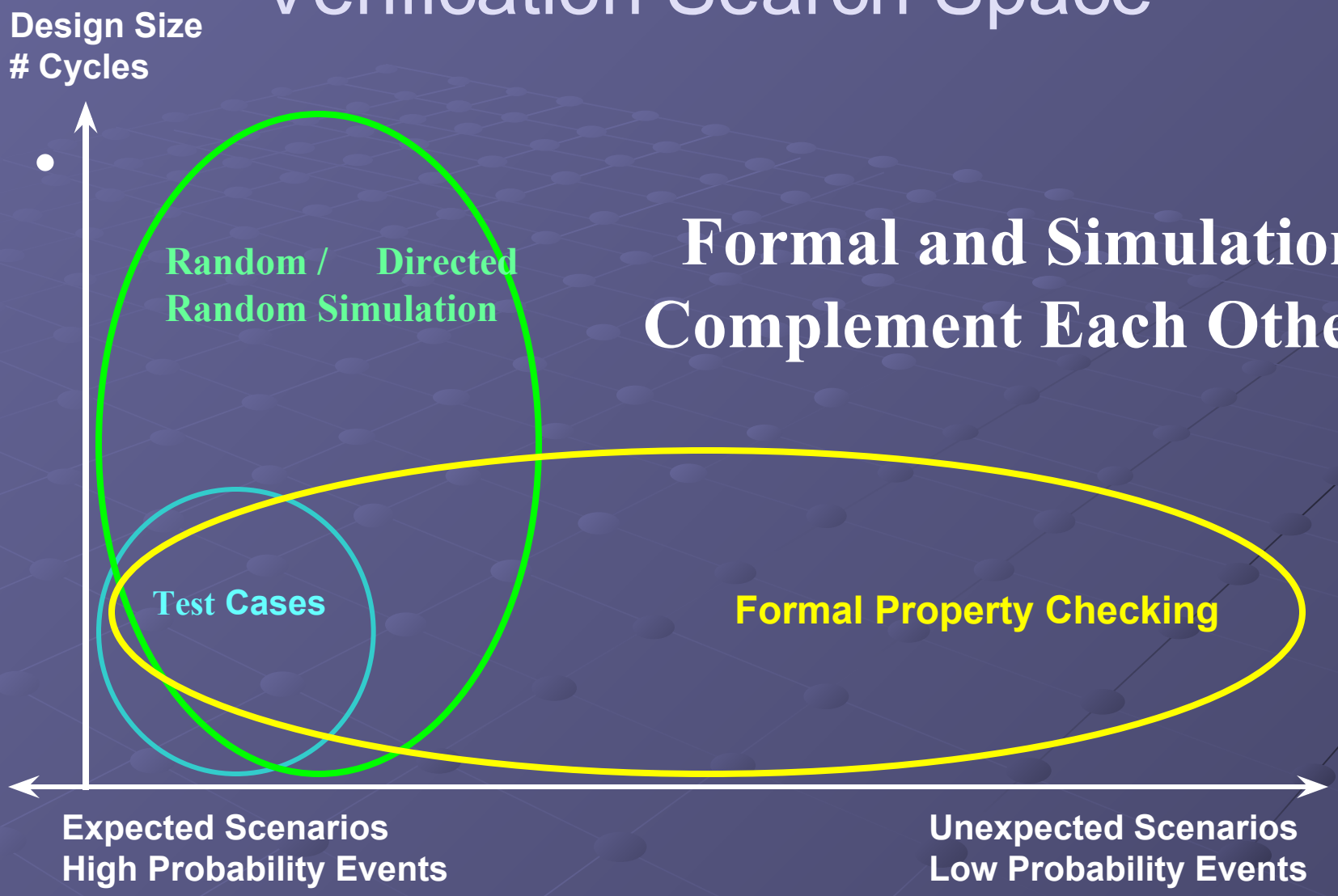
- Full Proof , searches all state space.
  - Highest level of verification completeness
  - Not always achievable.
- Bug Hunting , searches part of the space.
  - Over constrained.
  - Simulation guided.

Formal vs. Dynamic

- Formal
  - Exhaustive check
    - Per strength of tool.
  - Unpredictable runtime and resource consumption.
  - Need only input assumptions and output rules.
  - Capacity limitation
    - Per strength of tool.
    - May need to reduce DUV to fit.
  - X treated as all possible values

- Dynamic
  - Needs coverage
    - Never covered…
  - Predictable runtime and resource consumption.
  - Large testbench infrastructure / development effort.
  - No capacity limit.
  - Every tool propagates X differently
  - Higher level of abstraction

Verification Search Space



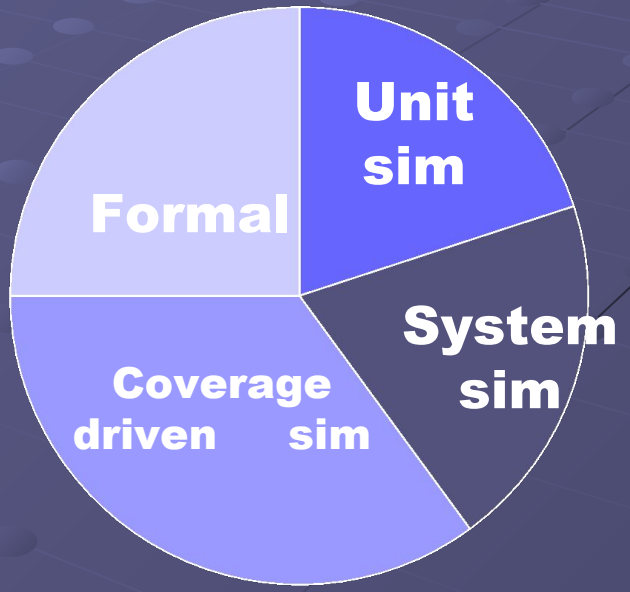
Formal and Simulation Complement Each Other!

What Type of Designs?

- Where simulation is weak
  - Arbiters , performance , QoS .
- Its hard to get confidence in Dynamic
  - Where probability of all events coinciding to exercise problem is remote
- Control logic
  - Bus interfaces, FIFO control, …
  - Not number crunching
- You don’t understand the design
  - Design too complex to be well understood
  - Architecture suspect – must test at level above arch!
  - Where you’re using a block the designer does not know closely.
- Reverse engineer legacy code

What weight give to Formal ?

- Simulation
- Acceleration/Emulation
- Formal
- FPGA



When do I verify?

- Block complete?
- Function complete?
- Is design going to change?
- Before/After dynamic.
  - Convergence
  - Additional quality
- Effort
- Achieve better quality in formal.

What level to test at?

- Lowest Level (Block)
  - best tool performance
- Highest Level (Full Chip/ Unit)
  - More meaningful proofs
  - Simpler assumptions on interfaces,
- Best: Best tradeoff between performance and simplicity.
- Corollary: How well the tool reduces the design is VERY important!

Planning and completeness

- Verification Plan
  - Part of Dynamic verification plan?
  - Separate verification plan?
  - Should we connect both?
    - Map block/function to feature.
- When is formal complete?
  - All outputs checked ?
  - All rules written?
  - Tools?

Properties and Assumptions Reuse

- Reuse Formal rules in dynamic.
- Reuse Dynamic assertions in formal.
- What to reuse ?
  - All rules?
  - Env assumptions?
- How to do it
  - Cut and paste?
- Dynamic env to Formal assumptions?
  - Can it be done?
  - Is it useful?

Who writes the assertions?

- According to Stuart Sutherland:
  - “Design engineers should write assertions to verify assumptions that affect the functionality of the design block.”
  - “Verification engineers should write assertions that verify design functionality meets the design specification.”

Who runs?

- Designer
  - Understands his white box better
- FV expert
  - Knows best how to use tools under complex circumstances.
- Multi disciplinary Verification Engineer.
  - Another tool in his portfolio.

Formal friendly design

- Guideline definition.
- How to push it.
- Do I want want to add another constraint to the designer?
  - In what priority?

Which language to use?

- PSL
- SVA
- Proprietary

- Standard.
- Best exploitation of tool.
- Usage in dynamic.