THE FORMAL VERIFICATION BOOK: PAST, PRESENT, AND FUTURE

ERIK SELIGMAN

NOVEMBER 2, 2016
OUTLINE

• Introduction
• FV Perception
• FV Goals
• FV Usage
• FV Effectiveness
• The Future
OUTLINE

• Introduction
  • FV Perception
  • FV Goals
  • FV Usage
  • FV Effectiveness
• The Future
• ASSUME (Scan_en == 1) ... 
  • Good thing we added cover points!
... And Another Classic Escape

- INVERSE (A,B) needed to prove C==0....
- But FEV mapping A -> B added too.
What did these have in common?

- Case 1: Vacuity!
- Case 2: Vacuity!
- Yet key user...
  - Made both mistakes, years apart
  - Solved differently
    - FPV: cover points / FEV: tool vacuity checks
  - Didn’t make connection (at first)
Core issue: Hacker Mentality

Does queue overflow?

Prove “assert property (!overflow)”

Done!

→ Missing Context/Big Picture
→ Missing Engineering Discipline
What Does a Strong FVer Do?

Astronomers
- Create upfront abstract hypothesis of how universe works.
- Reverse engineering is often required.

Dentists
- Regular checkups (regression)
- "proof-decay", Filling cavities during regression
- Maintain records for large population.

Wine makers
- Pruning, grafting, and genetic engineering (of RTL)

Inventors
- Need flexible proof systems that support building solutions
- hone checking recipe

Quilt makers
- Stitch together many small specifications

Environmentalists
- Reduce, Reuse, Recycle

Bungee jumpers
- Development cycle too long
- Need interactive, responsive proof systems.
Gaps in Big Picture Knowledge

• Perception: What is FV?
• Goals: When/Why to FV?
• Usage: How to FV?
• Effectiveness: How to FV Better?

➔ Need a book to gather & relate BKMs in these areas!
OUTLINE

• Introduction
• FV Perception
  • FV Goals
  • FV Usage
  • FV Effectiveness
• The Future
FV != Greek Letters

(*54·43. \forall \alpha, \beta \in 1. \mathcal{C} \cdot \alpha \cap \beta = \Lambda \implies \alpha \cup \beta \in 2

\textit{Dem.}

\forall \cdot 54·26. \mathcal{C} \cdot \alpha = \iota\langle x \cdot \beta = \iota\langle y \cdot \mathcal{C} \cdot \alpha \cup \beta \in 2 \implies x = y.

[\forall 51·231]

[\forall 13·12]

\forall (1) \cdot 11·11·35. \mathcal{C}

\forall (\iota\langle x, y \cdot \alpha = \iota\langle x \cdot \beta = \iota\langle y \cdot \mathcal{C} \cdot \alpha \cup \beta \in 2 \implies \alpha \cap \beta = \Lambda

\forall (2) \cdot 11·54. \forall 52·1. \mathcal{C} \cdot \text{Prop}

From this proposition it will follow, when arithmetical addition has been defined, that 1 + 1 = 2.

(snapshot from Russell & Whitehead, Principia Mathematica)
Is FV Just For Specialists? – NO!

• Old usage model:
  FV Dragons use FV Tools
  Everyone else uses sim, emu, …

• Modern view– FV is part of everyone’s toolkit
  • Every engineer uses the right tool at the right time
FV Is a Platform, Not Isolated Tools

- Control
- Regs
- Design
- Exercise
- CONN
- Full
- Proof
- RTL
- RTL
- Equivalence
- Bug
- Hunting
- RTL
- Netlist
- Equivalence
- ARCH
- Models
- Protocol
- Checking
FV Isn’t Extra Cost to Project

• Common view: FV = extra safety net for last mile
• Dangerous!
  • On one project, experienced FV team reallocated in “efficiency” drive

• FV (like all engineering) justified by *Return On Investment (ROI)*
FV: The Right Thing To Do

“FV wherever we can, simulate where we must”
OUTLINE

• Introduction
• FV Perception
• FV Goals
• FV Usage
• FV Effectiveness
• The Future
Does FV Need Goals?  -- YES

• What do you really want?
  • General Bug Hunting
  • Full Proofs
  • “Last Mile” Corner Cases
  • Specialized Apps

FV is smart– maybe we should do some before tapeout
FV = Normal Part of Validation Plan

<table>
<thead>
<tr>
<th>Unit</th>
<th>Logic Style</th>
<th>Unit Validation</th>
<th>Integration Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus Interface</td>
<td>Protocol</td>
<td>FPV</td>
<td>Formal CONN</td>
</tr>
<tr>
<td>Mem Controller</td>
<td>Control</td>
<td>FPV</td>
<td>Fullchip Sim</td>
</tr>
<tr>
<td>Signal Processor</td>
<td>Analog</td>
<td>Analog Sim</td>
<td>Fullchip Sim</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Leverage & Reuse FV Across Project Lifecycle

Formal Equivalence Verification (FEV)

Exploration / Architecture

Front End / RTL

Backend / Netlist

Silicon/PRQ

Formal Property Verification (FPV)
Is Full Formal Rigor Required?

• Ideal

• Maybe more practical (& exponentially better than sim):
OUTLINE

- Introduction
- FV Perception
- FV Goals
- FV Usage
- FV Effectiveness
- The Future
Apps – Are They Formal?

• Encourage users to understand context!
  • Debugging unexpected issues
  • Opportunities for slight variants: Control Regs ➔ Fuses?
  • Recognize opportunities across flows

I’m just using a control register tool
FV Workflow: What Professors Told You

Create Properties

Create Assumptions & Formal Env

Run Proofs

Pass?

All Done!

Fail?

Report Design Error
FV Workflow: Reality = “Wiggling”!

- Create Properties
- Create/Modify Assumptions & FormalEnv
- Run Proofs
- Pass?
- Maybe Done!
- Fail?
- Try to understand counter-examples
- Report Design Error

Really really sure failure is real?
What About Complexity?

- Know basic complexity methods + bug hunting
- ... And when to ask for help!
• Don’t panic if you can’t get full proofs
• Bounded proofs are exponentially better than simulation
Plan for ROI Measurement

• Did FV improve TTM & costs?  *Prove it!*
  • RTL 1.0 faster due to early exercise
  • Fewer engineer-hours for design & validation
  • Reduced late-stage “code churn”
  • …

• Need to carefully measure effort & results (not easy)
OUTLINE

• Introduction
• FV Perception
• FV Goals
• FV Usage
• FV Effectiveness
• The Future
Start with covers, not assertions

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion1</td>
<td>PASS</td>
</tr>
<tr>
<td>Assertion2</td>
<td>PASS</td>
</tr>
<tr>
<td>Assertion3</td>
<td>PASS</td>
</tr>
</tbody>
</table>

Assertions pass, I must be doing great!

Hmm, do these waves do what I want?
Encourage “Cheating”
Think About False Positives

• SVAs match intention?
• Expected behavior in waveforms?
• Vacuity checks?
• Division of labor?
• Assumptions reviewed & simulated?
• How/why did I “cheat”?
assign index = ...
A1: assert property (!$rose(req[index]));
Linting Remains Important (II)

assign index = ...
A1: assert property (!$rose(req[index]));

- Should the property **pass** or **fail** here?

- **FAIL**– SVA compares current req[1] to previous req[0]!

- Lint rule: flag cases where array index is sampled variable
“Easy” Things Can Kill You

Digital Logic

“Analog Block”
Coverage Is Critical

I am checking my RTL with simulation.

We better be careful about coverage.

I am checking my RTL with formal verification.

Great, I have nothing to worry about.

WRONG!
OUTLINE

• Introduction
• FV Perception
• FV Goals
• FV Usage
• FV Effectiveness
• The Future
FV = “Normal” Part of Toolkit

- Groundwork laid in college
  - Everyone takes Tom Schubert’s class!
    - ... or another class using our book

- FV ≠ separate category
  - Included in conference sessions on Front End, Validation, etc.

- Planning at all design/validation phases
- Every engineer knows basics
Formal Cooperation With Other Flows

• Non-FV tools share methods / collateral
  • Same build, debug for sim & FV
  • Specialized flows (power, scan, CR) prep FV inputs
  • VIP should support FV

• More cooperation for mixed-vendor envs
  • Can our vendors do a better job?
    • `sim_compile --guess_paths other_vendor.log`
  • Opportunity: better industry standards
    • Do different vendors truly need incompatible interfaces?
Tools Help Measure ROI

• Need more vendor support here
  • Will help sell more tools!
• Intelligently track user effort?
• Intelligently log bugs found?
• Track results across project phases?
• Synthesize results into charts comparable with simulation?
  • Be careful about naïve “Universal Coverage” integration of sim + formal
Better Tool Intelligence

• In engines, expect “new miracle every 5 years”
  • But SAT is NP-complete

• More hints for users?
  • Detect common fallacies
  • Pruning suggestions
  • Auto-ID relevant reusable collateral
  • …

• Continue to improve debug / root-cause
  • Remember, “Wiggling” = majority of engineer time
FV = PLATFORM, NOT TOOL

- Apps are nice… But hiding FV limits user learning
  - And kills ability to solve slightly related problems
- Need user visibility/extensibility
  - For both generated properties & app code
- Enable fine-grained customer control
  - Complex customization like C++/VPI in sim
- Provide flexible abstraction libraries
  - Where is UVM for Formal?

- Don’t sell us Apps—sell us a flexible, user-extensible system!
FURTHER READING

• Check out book at http://formalverificationbook.com!
  • Or read e-copy at Safari Books Online

• And to relax afterwards, read Erik’s “fun math” book, Math Mutation Classics.