Verification of Virtual Platform Models - What do we Mean with Good Enough?

Ola Dahl, Ericsson
Jakob Engblom, Intel
All models are wrong but some are useful

George E.P. Box

A model should be as simple as it can be but no simpler

— Albert Einstein —

https://www.lacan.upc.edu/admoreWeb/2018/05/all-models-are-wrong-but-some-are-useful-george-e-p-box/

https://www.azquotes.com/quote/531521
• Ola Dahl
  • Senior Specialist Model-Based Development
  • **Ericsson**, Stockholm, Sweden
  • Ericsson AI
  • Software Engineering
  • Control, Modeling, Signal Processing

• Jakob Engblom
  • Director Simulation Technology Ecosystem
  • **Intel**, Stockholm, Sweden
  • Intel® Simics® virtual platforms since 2002
  • Simulation, modeling
We don’t have all the answers... but hopefully some good questions
What Does Correctness Mean?
Correctness, Software Perspective

VP purpose: run, develop, test **software**

Software

Runs on

Virtual platform model

Good: Equivalent

Hardware

Runs on
Correctness, IP Block/Hardware Perspective

VP purpose: act as reference for hardware implementation

Virtual platform model

Good: Find Errors

Verification

Uses

Checks

Hardware
Too Much Correctness?

VP good enough for software development and test

Virtual platform model

VP good enough for hardware verification

Higher speed

More details

Typical “fast virtual platform”

Typical “golden reference model”

At what level of observation is the model correct?
Is the software virtual platform model the same as the hardware golden reference?
Note: (Software) Correctness: Is Being Forgiving Correct?

All software that runs on the hardware runs on the virtual platform. Good enough?
Specifications and Implementations
In a Perfect World: One Specification to Rule them All
Note: Single Spec – Multiple Implementations

For example, an instruction set specification... (ARM, RISC-V, X86, ...)

Multiple software stacks – different operating systems, compilers, ...

Ideally, any software runs on any VP or hardware
However...
“The Software Works on the Hardware”

Is this VP correct? What happens when the software or hardware is exchanged for a different implementation?
“My Hardware Implementation is the Spec”

The specification is derived from the actual hardware implementation

The software is also chasing specs from the hardware team

The VP is constantly having to chase specs from the hardware team

Happy: unconstrained, hacking away, changing the hardware implementation, and by implication the spec…

Following the spec does not mean you are correct vs hardware – specification updates are optional, late, and inconsistent
“The Software Works on the Virtual Platform”

The VP is implemented from the specification

Virtual platform model

Specification

Software

Hardware

The software is designed by testing on the VP – whatever the VP allows is OK

VP and software can go off on a tangent together... Unclear that the software works on hardware...
“The Hardware Said So”

VP might not match the specification and not run the software – the hardware might have specific interpretations of the spec, bugs, etc.
“Bug-Compatibility”

The VP models a particular hardware variant and revision – better remember that it deviates from spec
“That’s Not My Specification”

Multiple editions or versions of the specification are in use – the “hardware spec”, “the user manual”, “internal/external spec”, …
Does your flow go from specification to model?
System Integration / Testing

Ola’s part
System Integration - Concepts

Amount of module testing (for A)
Amount of module testing (for B)
Amount of integration testing (for C)
Software on a Virtual Platform (VP)
Some Words from Software (and Google)

who does unit tests and who does integration tests

About 349,000,000 results (0.66 seconds)

5. **Unit testing is performed by the developer. Integration testing is performed by the tester.** 30 Aug 2022

https://www.geeksforgeeks.org › difference-between-unit-...

**Difference between Unit Testing and Integration Testing**
Employ the principle of software unit testing to the TB [testbench] code early to minimize the age old “is it the DUT [Device under Test] or the TB that’s wrong?” debug cycle*

*The Challenges of Verifying an Arm CPU, Scott Kennedy, Arm, 2022 - [link](#)
Software on VP – Take 1

Software: my software test fails on your VP

Virtual platform: Oh, I'm sorry to hear that

Software: are you sure that the VP is Ok?

Virtual platform: yes, the VP works fine

Software: How do you know?

Virtual platform: I tested it

Software: How?

Virtual platform: I ran your software test*

*The software tests from an earlier version of the software
Software on VP – Take 2

my software test fails on your VP

yes, the software works fine

I tested it

I ran it on your VP*

Are you sure that the software is Ok?

How to you know?

How?

*The software ran on an earlier version of the virtual platform
Software on VP – Only Integration Tests

A  VP
B  SW
C  SW tests

- Amount of module testing
- Amount of module testing
- Amount of integration testing

A  VP testing
B  SW testing (no VP)
C  SW testing on VP
Borrowing – Key Concept

Each team \((A, B/C)\) uses a fixed baseline (a fixed version) of the other team's products.

The baseline is moved periodically (like once a week).
Hardware (IP, RTL) Verification

A  VP model of IP ("reference model")
B  RTL (for the IP)
C  UVM testbench

A  VP model testing
B  RTL testing (no testbench)
C  UVM testing
Hardware (IP, RTL) Verification

How do you know that your ref model is good enough?

I run it in the UVM test bench, where its outputs are compared with the outputs from RTL.
How do you know that your RTL is good enough?

I run it in the UVM test bench, where its outputs are compared with the outputs from the ref model.
Hardware (IP, RTL) Verification

A  VP model of IP ("reference model")
B  RTL (for the IP)
C  UVM testbench

A  VP model testing
B  RTL testing (no testbench)
C  UVM testing

Amount of module testing
Amount of module testing
Amount of integration testing
Borrowing, Again

Each team (A, B/C) uses a fixed baseline (a fixed version) of the other team's products

the baseline is moved periodically
This team gets a huge pile of tests... do they motivate their cost?

Tests from team (1 ... n)
Deliverable from team (1 ... n)
→ delivers to
← borrows from
Module Testing on Different Levels

How much should you test at each level?
Employ the principle of SW unit testing to the TB code early to minimize the age old is it the DUT or the TB that’s wrong?” debug cycle*

To what extent shall we "test the tests"?

*The Challenges of Verifying an Arm CPU, Scott Kennedy, Arm, 2022 - [link]
Module Testing – Is There a Scale From None To "Significant" To "Too Much"?
Stubs/Mockups/Verification IP/et cetera
System pre-integration test of A by borrowing from B

A  real product
B  real product
C  real product
System pre-integration test of **A** by borrowing (specs, concepts) from **B**

- **A** real product
- **S** stub
- **C** real product
System pre-integration test of A by borrowing (specs, concepts) from B and C

- A: real product
- S: stub
- T: stub
Borrowing with Stubs: Software and VP

A  VP  uses  T  Tests that represent (mimic) the SW that will run on the VP

C  SW testing on VP  uses  S  A simplified HW model e.g. built for host, that mimics the VP

*Each team (A, B/C) manages their own stubs*

*Breaks the borrowing cycle*
Borrowing with Stubs: VP and RTL

Each team (A, B/C) manages their own stubs

Breaks the borrowing cycle
Borrowing, Stubs, and Module Tests

• What is the right amount of borrowing?
• How much unit testing?
• How much integration testing?
• How much integration testing with stubs?
Do You Do the Right Amount of Unit Testing?

If not... too Little or too Much?
Different Kinds of Bugs

Still from Ola
implement → Doing the thing right

implement → Doing the right thing
Implement

Doing the thing right (DTR)

Implement

Doing the right thing (DRT)
Bugs

I did not implement what I intended to implement

I implemented what I intended to implement, but it was the wrong thing to implement

a DTR bug

a DRT bug
What if?

there are no DTR bugs in A
there are no DTR bugs in B
there are no DTR bugs in C

And/but the integrated system does not behave according to its spec (its tests fail)

how do we proceed?
is it the case that ...

if the subsystems being integrated into a composite system do not have any DTR bugs

and this is true also for the integration framework (the "wiring")

then the reason for the composite system failing can only be due to DRT bugs
noting that ...

DRT bugs occur due to different parties interpreting a spec differently

and hence that DRT bugs are not solved by "trouble-shooting" (rather by re-reading specs and discussing)

we might conclude that system integration becomes requirements management
Recall...
Imagine a Bug Report... That Looks Like This

Dear VP team,

I ran my test on your VP and it failed.

Here are instructions for how to reproduce

If needed, please contact us and we can set up a meeting for collaborative trouble-shooting
Dear VP team,

I ran my test on your VP and I saw an unexpected value in registers R1 and R2.

I expected these values (values mentioned), according to spec rev version 43, but I saw these values (values mentioned)

If needed, please contact us and we can set up a meeting for a requirements discussion (perhaps we have used different spec versions?)
Imagine a Bug Report... That Looks Like This

Dear VP team,

I ran my test on your VP and it failed.

Here are instructions for how to reproduce

If needed, please contact us and we can set up a meeting for collaborative trouble-shooting

and solved by debugging
Dear VP team,

I ran my test on your VP and I saw an unexpected value in registers R1 and R2.

I expected these values (values mentioned), according to spec rev version 43, but I saw these values (values mentioned)

If needed, please contact us and we can set up a meeting for a requirements discussion (perhaps we have used different spec versions?)

and solved by a feature update (which should be possible to estimate)
If we do more unit testing, and more integration testing with stubs?

Do we get a return on investment in the form of less trouble-shooting? (fewer DTR bugs, perhaps as a vision: only DRT bugs)
Some “Software Laws”
Conway’s Law

• The organizational structure will be reflected in the structure of hardware and software

• If major interfaces in the VP do not follow the organizational structure, problems will develop over time

• Be aware: Organizational boundaries can limit the access to specifications, limiting or complicating VP modeling
  • Corollary: to model something, you must know someone in the org building it
Hyrum’s Law

• *Developers using an interface will likely come to rely on undocumented or unspecified behaviors*

• The VP model cannot be expected to follow the Hyrum’s law aspects of the hardware

• Rely on the **specification** and consider issues as software or hardware bugs (in the case implementation aspects creep in)
Goodhart’s Law

- If a measure becomes a target, the measure becomes pointless
- Don’t make volume or delivery dates of platforms into targets
- That will distort the process
Summary
Main Points

- Define “correctness” appropriately
- The specification should be king
- Unit tests are key to successful integration
- Consider what a test actually tests
- Doing the right thing, or doing the thing right?
- Organization matters
The End!
The Only way to Know is to Test