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

Tutorial type: Industrial.

Speakers:

- Jakob Engblom, Intel, Stockholm, Sweden. Jakob.engblom@intel.com, +46734368958
- Ola Dahl, Ericsson, Stockholm, Sweden, ola.dahl@ericsson.com, +46725838355

Introduction

The use of virtual platforms (VP) for software development and system integration is well-established in industry. Different companies use different tools, frameworks, and technologies, but despite this there are many common patterns and anti-patterns in how models get developed, deployed, and debated. In this tutorial, we would like to share from our experience around the verification and validation of virtual platform models from inside of large enterprises.

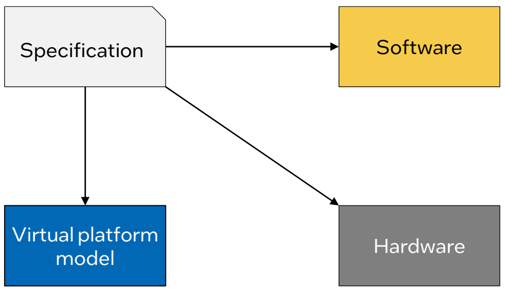
Summary of Contents

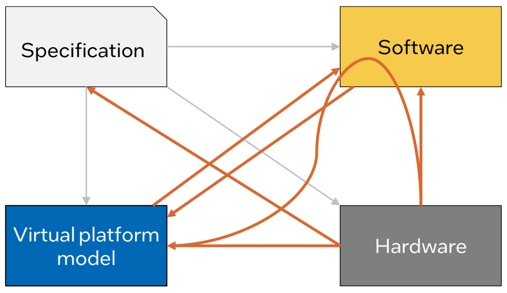
When people ask if a VP model is correct, a VP developer may answer, perhaps with reference to one of their more famous role models, that "a model needs to be good (accurate) enough, for its purpose". But how good is good enough, and from what perspective?

The Four Entities

There are four entities involved in the creation of a virtual platform model. The **specification** of the design, the implementation of the **hardware** (that the model is modeling), the **virtual platform** model itself, and the **software**. The software is designed to run on the hardware. The software should also run on the VP models, and all should be derived from the specification.

However, there are several anti-patterns to this discipline. For example, the VP model might have been based on software that was written based on observed hardware behavior. See Conway's law below.





Some anti-patterns in VP and hardware development

Virtual Platforms and System Integration

When integrating two or more subsystems into a system, it is desirable to check that the integrated system works as expected. An important trade-off is just how much the subsystems be tested before they are integrated together, and how much of the testing can left to the integration?

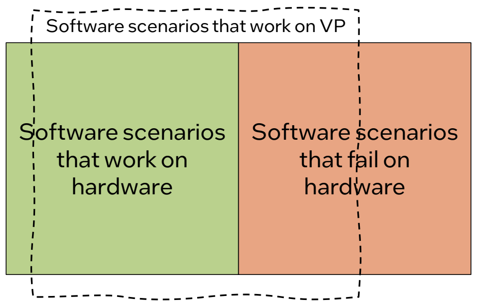
When running software on a VP, for example, a VP developer may argue that "the model will eventually meet the software, and then we will see if it works". So, not much testing is needed on the model in isolation.

Another, more test-ambitious VP developer may say that "I will test my virtual platform model and ensure that it is implemented according to my interpretation of the specification, before it runs software”. If the integration test does not behave as expected, the virtual platform model developer can claim that the fault lies within the software. If the software developer had done a similar testing, the software developer can claim that the fault lies within the virtual platform model.

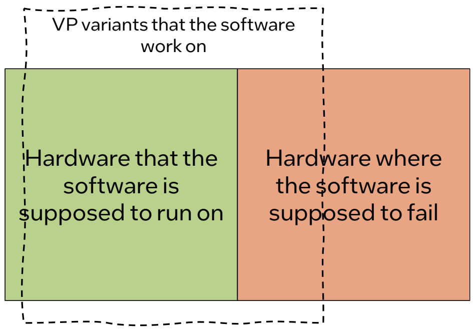
In the tutorial, we will continue this reasoning on how to resolve the conflict of “where the fault lies” and the need to apply tests at all levels of integration.

Negative and Positive Testing

VPs need to provide the ability to test and detect failure scenarios in addition to happy-case tests. That a software stack boots on a VP says very little about the quality of the VP or the software. Software is often far too forgiving and will run despite the VP setup not really matching reality. Or the VP model is too forgiving versus the software, allowing incorrect software to run. We need models and tests that take both positive and negative into account.


Software scenarios that work on VP
Software scenarios that work on hardware
Software scenarios that fail on hardware

VP is not strict about negative scenarios


VP variants that the software work on
Hardware that the software is supposed to run on
Hardware where the software is supposed to fail

Software adapts to the VP model

Software Laws

Conway’s Law: The organizational structure will be reflected in hardware and software. If major interfaces in the VP do not follow the organizational structure, problems will develop. Organizational boundaries can also limit the access to specifications, limiting or complicating VP modeling.

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 – they have to rely specification and consider issues as software or hardware bugs in case implementation aspects creep in.