Shifting Left Together

Shifting Left Together

Enabling the Ecosystem with Virtual Platforms

Jakob Engblom
Intel
Stockholm, Sweden
jakob.engblom@intel.com

***Abstract*—"Shift left" is the practice of breaking dependency chains in hardware and software development, so that tasks can start earlier and run in parallel. The goal is to shorten the time to market and have software ready with support for new hardware features when the hardware arrives. Inside silicon companies, it typically means using methodologies like virtual platforms, environment models, emulators, FPGA prototypes, and hybrids thereof to allow firmware, boot code, drivers, operating systems, and application code to be developed, tested, and integrated before the chips arrive. However, shift-left cannot just be internal to a chip vendor – it is necessary to enable the ecosystem in the pre-silicon phase in order to build a complete product.**

In this paper, we will look at pre-silicon software and system development using virtual platforms in an ecosystem. There are technical issues like building and delivering models, as well as project aspects like how to rejig projects to shift left. What has to be modeled – a reference platform or an actual customer system? What happens once hardware arrive? How is the virtual platform leveraged for system integration, including other system components and existing hardware generations? How do they fit with existing workflows?

The introduction of virtual platforms can also enable deeper changes to how product development, integration, and testing are done – in particular across generations.

Keywords—simulation, virtual platform, shift left

I. Introduction

Computer hardware is pretty much useless without software. When a new processor, platform, System-on-Chip (SoC), or system is launched, software has to be developed and updated to support the new hardware. This includes basic boot code, BIOS (Basic Input-Output Systems), and drivers – as well as OS (operating system) kernels, OS services, programming libraries, and application programs. Over time, this has become more and more important – as the impetus for new SoCs has shifted from just performance improvements to the whole set of features provided, software becomes a bigger and bigger part of the "launch package" for a new computer architecture, chip, platform, or product.

Once upon a time, software support for a new computer generation was only developed and tested after the first

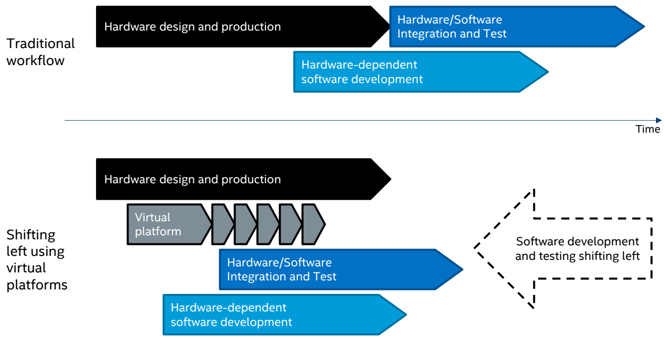

Figure 1. Shift-left concept

hardware arrived. Software would be developed using various forms of physical prototypes, in parallel to the finalization, stabilization, and volume production of the hardware. Such a flow creates huge schedule risk, and puts enormous stress and pressure on the software developers who have to work under high time pressure to get their pieces done so that the product can ship.

To reduce the stress, you get practices like "coding in the dark" – developing hardware drivers without hardware to test on, just coding in register offsets and bit fields from the spec sheets and hoping that things are right. Testing is still not possible until the hardware arrives… and usually results in long nights and plenty of overtime.

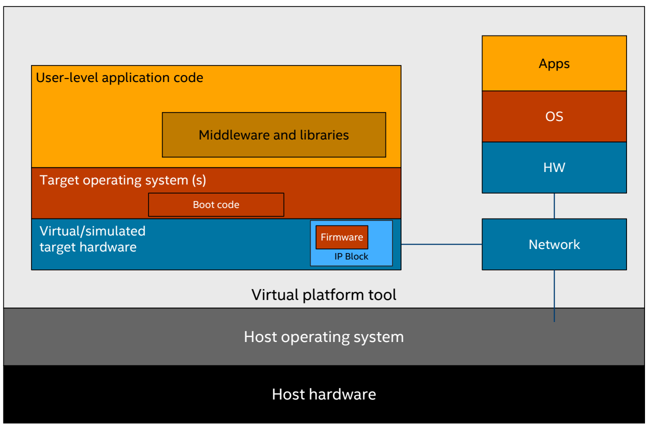

Figure 2. A virtual platform

A better solution is to use some kind of simulator or virtual platform to model the upcoming hardware - providing software developers with a platform to run and test software on before the hardware arrives. This makes it possible to develop and test software in parallel to the hardware development, as illustrated in Figure 1. The virtual platform changes to keep up with hardware specification changes, making sure that software can be tested against the current specification [1][2][3][4][5][6][7].

A virtual platform (VP) is a software program that simulates the hardware of a computer system such that the unmodified software for the real hardware can run just like it would on the real hardware. As shown in Figure 2, virtual platforms can cover everything from individual intellectual property (IP) blocks with firmware to networks of multiple machines each running their own operating system. The key is that software that will eventually run on the final hardware platform can be run, tested, and debugged on the virtual platform.

Using computers to simulate future computers in this way is not a new idea by any means – its use is documented in several projects as early as the 1960s [8][9][10]. However, with a few exceptions, the technology has been confined within the organization building a chip or platform. Internal simulation tools are rarely robust enough to put in the hands of customers, and customers have been hesitant to invest in the effort needed to actually make good use of such tools. It is also the case that the main effort of supporting new chips and platforms fall on the silicon supplier.

The landscape is changing, as tools mature and the benefits of developing software early become apparent to original equipment manufacturers (OEMs) building upon chips and platforms from silicon providers.

In the rest of this paper, we will take a look at some of the mechanics, observations, and best practices we have observed doing shift-left (as illustrated in Figure 1) over the years. At Intel, we are using the Wind River Simics® [1] virtual platform to enable OEM customers early, and pull them along to the left.

Note that this paper does not address the use of virtual platforms to architect and design hardware – we are looking at the platform enabling use case, about how to make sure the software is available once the hardware arrives.

II. Internal Silicon-Vendor Shift-Left

Internal shift-left at silicon vendors is standard practice today. Virtual platforms developed using transaction-level models (TLM) [1][2][5][6] along with fast functional instruction-set simulators provide a way to develop all kinds of software – from firmware inside of IP blocks [11], to boot code and BIOS [12], to OS and drivers [3][6], to compilers, performance analyzers, and debugging tools.

Depending on the scope and length of the silicon project, virtual platforms can be provided many quarters or even years in advance of first hardware [7][12]. Silicon developers often combine VPs with other tools like RTL (register-transfer-level) simulators, emulators [12] and FPGA (Field-Programmable Gate Array) prototypes [13] to validate the actual hardware implementation against software – and the software against the hardware implementation. This finds errors early, reducing the risk of having to "re-spin" and bug-fix the hardware.

Internal shift-left is facilitated by the naturally low barriers between teams and groups inside a company – confidentiality and NDAs (non-disclosure agreements) is not a concern, and quick hacks can be shared with the understanding that they are not permanent. The colleagues building the hardware are just a phone call or quick instant message (IM) away from the virtual platform team, and the software teams can reach out to either.

When the VP models are shipped to external users, a bit more structure is called for than for pure internal use. This is probably why it is less common to see VPs used for external enabling than for internal acceleration.

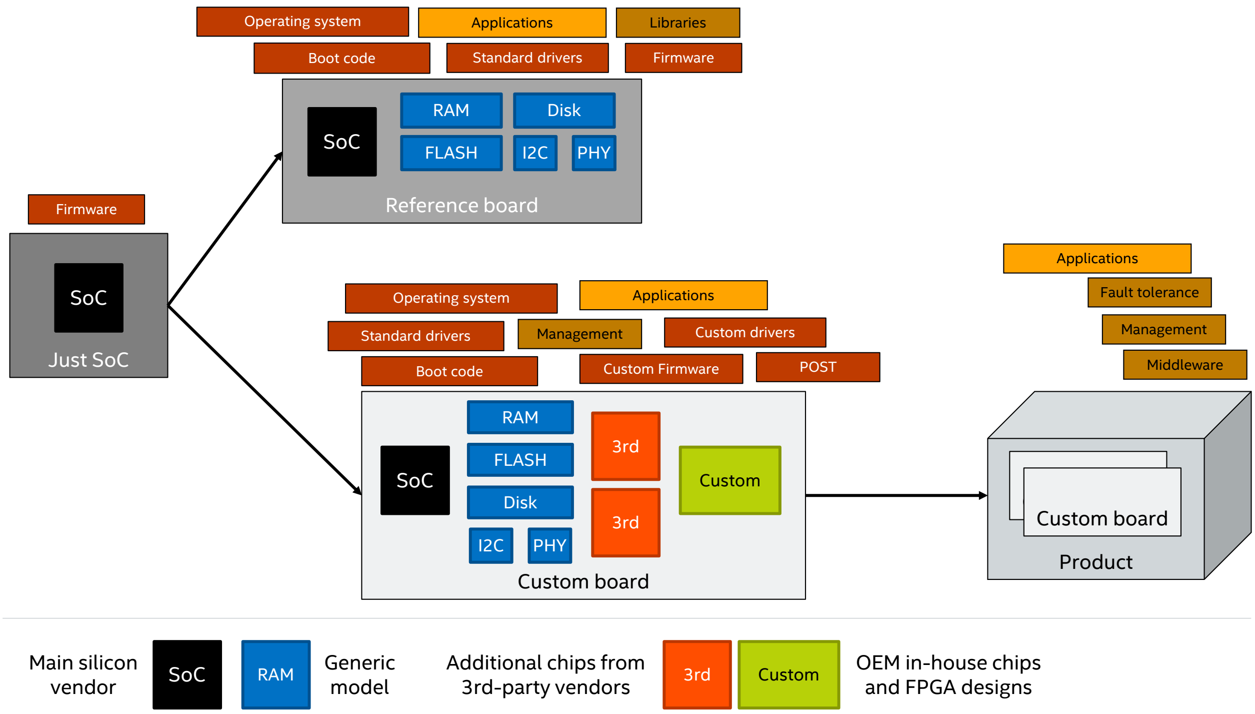

Figure 3. Variants of virtual platforms and the software that can be developed and tested on them

III. WHAT IS IN A VIRTUAL PLATFORM?

The scope for a virtual platform varies over time and between use cases. A rough guide is shown in Figure 3. The starting point is a VP for "just" the core SoC, which is suitable for firmware development. Adding generic external parts like Random-Access Memory (RAM), FLASH memory, and disks, gets us to a virtual *reference board*. Such virtual reference boards often include small additional components like I^2C temperature sensors and memories, and network physical interfaces (PHYs).

The reference board is sufficient to develop platform support for standard operating systems as well as drivers and applications that take advantage of the new platform. Such a reference board is typically what is used inside a silicon vendor, and they provide enough details to allow the boot and test of commercial and open-source operating systems.

For example, for a server system, a reference platform would cover the server processor SoC and supporting chips (usually known as the platform controller hub, PCH) along with the core chipset, RAM dual-inline memory modules (DIMMs), disks, boot flash, and some simple models for I^2C pieces like temperature sensors and power control. Such a reference board allows the execution of benchmarks, including networked database benchmarks and web servers [14].

Real products are not built from reference boards, however. *Custom boards* are designed by taking the basic SoC and platform from the reference and adding new components. Typically, this includes both 3rd-party hardware components and in-house custom hardware, on a custom-designed printed circuit board (PCB). One or more of the custom boards are then packaged into the final product that is sold to customers.

A model of a custom board can be used to develop not just the software specific for the platform from the silicon vendor, but also code specific to the custom board and vendor. Such software include system management code, power-on and self-test (POST) code, customized firmware running on the main platform and 3rd party and custom chips, etc.

Continuing to the full product model, even more software can be addressed by the virtual platform, including functionality like fault tolerance, communications middleware, management across and between boards, and system-level applications [1][2][16][17].

Continuing on the server example, a custom board would include the Board Module Controller (BMC) chip (and its custom firmware), Peripheral Controller Interface Express (PCIe) switches, serial attached SCSI (SAS) disk controllers, as well as custom chips that add value beyond a generic server. The whole product would encompass a rack of boards, and allow the testing the management of boards over BMC backplanes, fault detection and isolation, networking between boards, and other such system-level functionality.

IV. SILICON CUSTOMER SHIFT-LEFT

A. Initial Engagement: Reference Platform

When silicon companies start to shift their customers (OEMs) left, it is typical to start with a reference board, as shown in Figure 3. Such a reference model serves to get the customer started and supports the most critical software development. It provides an executable specification of the new platform (SoC) to the OEM for initial software porting.

For example, if an OEM has its own BIOS, porting the BIOS to the future platform can be done on the reference platform. The same goes for platform drivers in operating systems, and the adoption of new instructions in software stacks. In the past, such work was often performed on early physical prototype boards and reference boards made available in small volumes by the silicon vendors.

However, physical prototypes only arrive after first hardware has been successfully manufactured and booted by the silicon vendor. The virtual platform arrives many months (or even a year or more) ahead of the physical platform, and allows the software porting work to be performed before any silicon is available. Once the first silicon arrives, it basically serves to test and resolve issues related to pure hardware issues as well as platform aspects below the level of abstraction of the virtual platform.

Experience indicates that compared to doing the all the bring-up work on hardware, developing and testing low-level code like boot code, BIOS, and drivers on a virtual platform before the hardware arrives cuts the time to successful boot on first hardware from months or weeks to days or even hours [3].

One particular aspect that is worth mentioning is the "blame assignment" problem. When early code does not work on early hardware, untangling where the issue actually lies can be a challenge. However, using the VP, it is trivial to get a trace of all software writes to the hardware. This trace can then be compared to the specification of the hardware, making it easy to see if the software design or the hardware design was at fault. This can save many roundtrips compared to working only on prototype hardware—when the hardware is unstable, telling software and hardware issues apart is usually incredibly challenging [15].

B. Further Engagement: Custom Platform

Once the effectiveness of the pre-silicon platform reference board is seen, customers typically realize that there is more that could be done in pre-silicon. If more components and parts of their actual custom board(s) can be made available as VP models in the pre-silicon time-frame, it becomes possible to develop and test more code before the hardware arrives. This takes shift-left to another level.

Moving to a custom (virtual) platform brings many benefits – as seen in Figure 3, it makes it possible to test much more of the differentiators and value-add of the OEM. Today, the core platform of a product is common to a lot of vendors, with product differentiation coming from the parts that are added to the base platform, both hardware and software. The integration of these pieces with each other and with the software stack is critical, and doing that integration in pre-silicon is tremendously helpful to ensure that the product hits the market on schedule and with the features and quality planned.

The development of a custom board model often starts small, adding models of simple items like I^2C-connected sensors that can rather easily mocked up and added to the reference platform. It might be adding memory configurations or disk setups that more closely match the actual custom platform. In the end, it becomes a matter of modeling OEM-specific hardware as well as specific chips and other components bought from third parties.

C. Ultimately, the Product

Going beyond the individual boards, the next step is to model the product itself. In some cases like a piece of consumer electronics, this might just be an enclosure around a board. In other cases, we are looking at racks and systems with many different types of boards [1][6][16]. For IoT, embedded, and control systems, the simulation can grow to encompass not just the computer platform and board, but also the physical system that is being controlled by the computer. In addition, it usually necessary to model the environment in which it operates [17].

Taking shift-left to its logical conclusion, at some point we are looking at a totally transformed product development flow that includes physical design, design-for-manufacturing, and post-deployment predictive maintenance using digital twins. The humble virtual platform for a silicon chip becomes a small but vital component in a much bigger whole.

V. Building a Custom Platform

To realize the benefits of a virtual platform of a custom board and product, it is necessary to build the virtual platform. Typically, this is not a job that the silicon vendor can or should take on – the OEM is likely not happy to share details of their custom chips and value-add with the silicon vendor that sells to its competitors. The third-party hardware vendors have similar competitive concerns. Finally, the silicon vendor really does not consider it its job to model hardware from external vendors – it wants its modeling teams to be deployed for shift-left of its own future platforms.

A. OEM Custom Hardware

Models for custom hardware is best built by the OEM itself, or using consultants that sit on site and in their networks.

A rough guide is that a modeling team at an OEM should have at least two members working full-time on modeling and virtual platform integration for a few years to make sense – otherwise, it will be hard to build up modeling proficiency and maintain it over time. Using third-party consultants make sense if this internal critical mass cannot be reached.

In the end, if the virtual platform deployment and integration is successful, it is not uncommon to see teams of tens of people building models, supporting hundreds of users and thousands of automated test runs.

If the custom chips are in active development, it is likely that there is already a model in place somewhere in the hardware development organization. Such models can sometimes be used to build fast virtual platforms – it really depends on the level of abstraction used. Our experience is that pretty much any model can be integrated into a complete custom platform model – in the end, a VP is just software, and there is always some way to make one piece of software talk to another piece of software. However, not all models are suitable for VP use. Detailed architecture models are typically too slow to work well as a software execution and integration vehicle.

B. Third-Party Hardware

Models for third-party hardware offers an interesting case. Sometimes, the chip vendors already have VP models to offer. Such models are often available for individual IP blocks sold to chip designers, but they are rather rare for complete chips. It is particularly rare to find models that are built to be integrated into something bigger unless that was the original idea.

Thus, the practical choice for the OEM is often to model third-party hardware itself, or employing a consultant to do so in the same way that internal custom hardware is modeled. Since the OEM is already a customer for the third-party hardware there tends to be a motivation to support the effort on the part of the hardware vendor. Also, if the model is only used internally at the OEM, the risk of inadvertent disclosure of secrets to competitors or unauthorized use of the model is rather small.

C. Platform Integration

The integration of all the virtual platform pieces into a custom platform is also something that should happen inside the OEM organization or in a dedicated services organization serving the OEM. It is critical to reduce the number of steps and organizations involved in the integration and testing work in order to move quickly and deliver new VP versions in the shortest time possible.

For fast-moving pre-silicon hardware, building a chain of providers that each add a small piece to the overall platform is a bad idea. New VP versions and model versions need to be integrated and bug fixes delivered to the end users within days or a week, and this is not easy to achieve with a long a chain of organizations. It is better to concentrate the integration to a single point that is responsible for bringing everything together.

VI. Moving target

Going back to the deployment of pre-silicon models of future hardware from a silicon vendor to an OEM, one factor that is sometimes overlooked is that the virtual platform provides a "moving target" in terms of the hardware that is

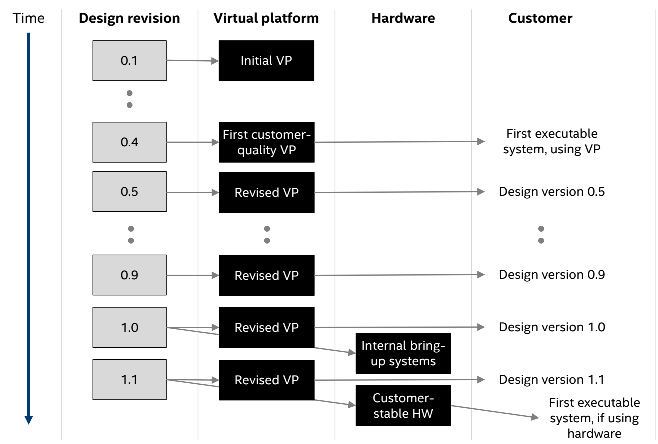

Figure 4. Virtual platforms provide a stream of design revisions

being modeled. As shown in Figure 4, the hardware design will change as the project progresses.

Changes can be big or small: blocks might be added or deleted from the SoC. System memory maps and register maps in blocks will be subject to change. Functionality in a block changes over time as the design and understanding of the problem and software matures. Changes can be expected until the point in time where the hardware design and implementation is frozen and the SoC “tapes in” and is sent for manufacturing.

Figure 4 also shows virtual platform drops beginning some time into a project – this is pretty typical, since the customer and the vendor will both want to have a reasonably stable platform before starting to share it. When the hardware starts to arrive, the first iteration of the hardware is usually used for internal bring-up, and there is often a small hardware revision (shown as “Version 1.1” in Figure 4) before the first customer samples or prototypes of the new SoC goes out.

A. *Keeping up with Changes*

When engineers first encounter shift-left and pre-silicon software and system development, one not uncommon reaction is that it is a hassle and annoyance to continuously update software to keep up with the changes to the hardware. Unfortunately, shift-left does not simply mean that you get the final hardware, only earlier. It means joining a process a working closely with the silicon vendor – just as if this was an internal chip being develop in a different team. It brings the vendor and customer together.

Put in a different way, joining shift-left is like signing up to pre-release versions of software and being an alpha tester of a new service – change is to be expected.

B. *Causing Changes*

Shift-left naturally brings along closer collaboration between the silicon vendor and the OEM customer. One benefit that this brings is that the customer can provide feedback on the design before it is frozen.

If first customer exposure to a design happens when the first hardware arrives, it is too late to do much with the current product. Feedback, shortcomings, and design problems will have to be addressed into a future revision or next-generation of the hardware platform. If the issues are identified in the pre-silicon time frame, they can be resolved before the silicon is frozen and be part of the initial silicon release.

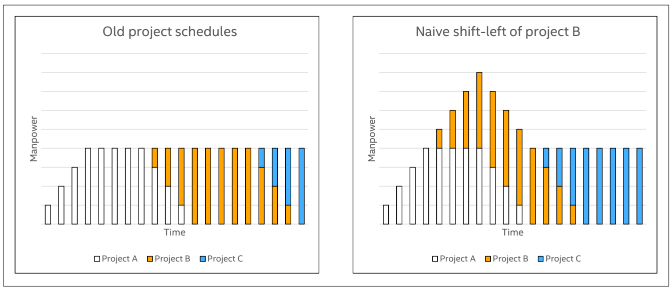

Figure 5. Over-simplified sketch of the planning impact of shift-left

Thus, shift-left makes for products that better suit the OEM customers as well as the silicon vendor, and that really work with the software and in the systems where it is intended to go.

On a more detailed level, virtual platforms enable communication between hardware designers and software designers before the final design is finalized. This can avoid software developer grief caused by poor hardware programming interfaces or mistakes in the design. There are many stories about poor hardware designs that would have benefitted from a software review before being finalized [18].

VII. Planning and Shift-Left

As discussed above, shifting left has huge benefits in terms of time-to-market and getting products done earlier with higher quality. However, it is not always trivial to shift development left, especially not the first time it is to be done.

Figure 5 shows a simplified sketch of what can happen: we assume that the organization has a certain pool of software developers and other resources that are carefully scheduled on project after project. If you naively shift project B left, the total load will essentially double for a while. That is not likely to work very well – thus, shifting left is a gradual process happening over several consecutive projects.

Of course, if developers were already developing code for the new platform without any access to test platforms or using various forms of approximation, this does not apply and the likely result is that the overall effort will go down as more testing can actually be done before the hardware arrives.

Still, the shift to shift-left has to be planned to avoid creating undue stress on the organization.

VIII. Continuing Right

Once shift-left is established as a way of working with new generations of hardware products, a stream of pre-silicon VPs will be appear as new generations of silicon platforms and related products are produced. The VPs are incredibly useful in the pre-silicon and pre-hardware timeframe… but they are just as useful after the hardware has arrived. Considering the VP only as a stop-gap before the physical platform arrives does not realize its full value.

In pre-silicon, the virtual platform should be used not just for manual testing and debug, but also for automated testing and in continuous integration (CI). This is especially important given the steady drop of new versions of hardware – an automatic workflow is necessary to verify that the software is indeed updated to work on the new hardware releases. If the customer is building a custom virtual platform (as discussed above), it is also necessary to validate the integration of different virtual platform models.

Ending the automatic testing and integration once hardware arrives is not a good idea – given that the product will most likely need ongoing software updates, it makes sense to keep the VP-based CI flow going even in post-silicon. Indeed, the software testing value of virtual platforms, especially those that model specific custom hardware and those that are integrated with external simulators, is independent of hardware availability [1][17][17].

Over time, as multiple generations and variants of hardware becomes available as in virtual platform form, virtual platforms can also be used to expand testing to include system-level interactions like the mixing of multiple generations of boards in rack-based and distributed systems and validating that updates to cross-platform software and middleware actually work across multiple platforms.

IX. Integration, Delivery, and Support

As discussed above, someone has to be responsible for integrating, delivering, and maintaining the complete VP to the end users inside the OEM. Ideally, such a team is also responsible for first-line support on the platform, as well as the integration of the VP into the overall development environment and automation flows. The team doing the integration work needs to be internal to the OEM so that it can respond quickly to users and have access to as many internal resources as possible.

In terms of practical delivery, a complete custom virtual platform can be daunting to configure and keep updated for individual users. Some kind of solution needs to be created that allows users to get started without becoming experts in setting up the simulator.

Simple solutions include using shared installations on servers that users access over networked file systems. The complete simulation setup can also be checked into version control (provided version control supports gigabytes of binaries), or packaged up in zip files. Yet another option is to use container technologies like Docker* to encapsulate and deliver virtual platform installations to users. A more ambitious solution is to build internal cloud services that allow the virtual platform to be invoked from internal web pages, hiding the details of the VP integration from users entirely [3]. Integrating virtual platforms into build and continuous integration flows is one extreme of this, where the VP is potentially not even directly visible at all.

When providing access to the virtual platform, one aspect that is important to consider is just how to efficiently deploy and test the software that users create on the virtual platform. The disk images and software images can sometimes be very large (like complete database setups or bootable enterprise Linux* images), and data movement might be a constraining factor. Should the VP come to the code, or the code come to the VP?

The key is always to reduce the amount of work an individual user needs to do in order to get access to the tools and models, to maximize the time spent doing useful work.

Since the virtual platform is "just software" it is easy to combine it with other software like firmware, boot code, operating systems, and disk images into ready-to-use setups that provide downstream teams with all they need to get going – without requiring each user to ensure that they have a consistent setup. The VP delivery to end users usually has to grow to include such critical software.

Using virtual platform checkpointing [1][19] it is also possible to optimize the workflow by providing booted-up and ready-to-use configurations to users. Such "nighly boot" workflows follow up the nightly build of a software platform with a boot up of a few important variants, and distributing the booted platforms to software developers.

X. Summary

"Shift-left" is the practice of moving software development, test, debug, and integration earlier in the time plan for silicon development projects. Using virtual platforms and other execution vehicles, software can be run and tested pre-silicon. Shift-left ensures that critical software support is in place when the hardware ships, ensuring that new hardware features are indeed turned into user benefits.

Traditionally, shift-left has been performed inside of silicon providers. By extending shift-left to the customers of the silicon providers, more software and product development can be done pre-silicon. This results in better products with higher software quality, along with a reduction in the time-to-market.

Extended shift-left starts with shipping reference virtual platforms for new silicon products to the silicon customers, and then evolves into virtual platform models of the actual customer board and product designs. In the end, the complete models might include models of the physical world and digital twins of in-production cyber-physical systems.

The change to a shift-left methodology requires some investment. It will change how development projects are scheduled and planned. There will need to be a support organization in place for developing, managing, and maintaining pre-silicon virtual platforms. To be maximally effective, the virtual platforms will have to be integrated into build, test, and continuous integration workflows – taking the place of hardware in pre-silicon, and augmenting hardware availability post-silicon.

Shifting left is the way of the future, come shift left with us!

References

[1] Daniel Aarno and Jakob Engblom, *Software and System Development using Virtual Platforms – Full System Simulation with Wind River Simics*, Morgan Kaufmann Publishers, 2014.

[2] Rainer Leupers, Olivier Temam (editors). *Processor and System-on-Chip Simulation.* Springer, 2010.

[3] S. Koerner, C. Werner, T. Hess, P. Schulz, M. Strasser, S. Wagner, H. Böhm, M. Troester, D. Bolte, H. Elfering, T. Pohl, K. Theurich, W. H. Miller, and P. Szwed. "Firmware verification and simulation in IBM zEnterprise 196", *IBM Journal of Research and Development*, Volume 56, Issue 1/2, January-March 2012.

[4] Claude Helmstetter, Jérôme Cornet, Bruno Galiléey, Matthieu Moy, and Pascal Vivet, "Fast and Accurate TLM Simulations using Temporal Decoupling for FIFO-based Communications", *Design, Automation, and Test in Europe (DATE)*, March 2013.

[5] Cornet, J., Maraninchi, F., Maillet-Contoz, L. "A method for the efficient development of timed and untimed transaction-level models of systems-on-chip." *DATE '08: Proceedings of the conference on Design, automation and test in Europe, pp. 9–14,* March 2008.

[6] Ola Dahl, Michael Lebert, and Eric Frejd, "TLM-based Virtual Platforms at Ericsson - Challenges and Experiences", *Proceedings of the Design and Verification Conference (DVCon) Europe 2016*, München, Germany, October 2016

[7] Tom de Schutter (editor), Better Software. Faster! Virtual prototyping best practices. Synopsys Press, 2014.

[8] D. P. Rozenberg, L. Conway, R. Riekert, *ACS Simulation Technique*, IBM Internal Memo, Mar. 15, 1966. http://ai.eecs.umich.edu/people/conway/ACS/Archive/ACSarchive.html

[9] Fuchi, K., Tanaka, H., Manago, Y., Yuba, T. "A program simulator by partial interpretation," *SOSP '69: Proceedings of the second symposium on Operating systems principles, pp. 97–104,* October 1969.

[10] David Mindell, *Digital Apollo*, MIT Press, 2008.

[11] Rocco Jonack and Juan Lara Ambel. "VP Performance Optimization – How to Analyze and Optimize the Speed of SystemC Models", *Proceedings of the Design and Verification Conference Europe (DVCON Europe)*, München, 14-15 October 2014.

[12] Carbonari, S. "Using Virtual Platforms for BIOS Development and Validation", *Intel Technology Journal*, Volume 17, Issue 2, 2013.

[13] Daniel Nenni and Don Dingee, *Prototypical - The Emergence of FPGA-Based Prototyping for SoC Design*, Semiwiki, 2016. http://www.s2cinc.com/resource-library/prototyping-book

[14] Jakob Engblom, "Running Large Software on Wind River® Simics® Virtual Platforms, Then and Now", Intel Developer Zone blog post, March 2018. https://software.intel.com/en-us/blogs/2018/03/15/software-on-wind-river-simics-virtual-platforms-then-and-now

[15] Condon, J., Kernighan, B., and Thompson, K.: *Experience with the Mergenthaler Linotron 202 Phototypesetter, or, How we Spent our Summer Vacation*, Bell Laboratories Technical Memorandum 80-1270-1, January, 1980.

[16] Jakob Engblom, "Internet of Things (IoT) in the Lab - Staging and Testing for the Real World", *Embedded World 2016 Congress*, Nürnberg, Germany, February 23, 2016.

[17] Jakob Engblom, "Continuous Integration for Embedded Systems using Simulation", *Embedded World 2015 Congress*, Nürnberg, Germany, February 24, 2015.

[18] Jakob Engblom, "Iterative Hardware-Software Interface Design", Wind River Blog, 22 December, 2010. http://blogs.windriver.com/engblom/2010/12/iterating-the-hardware-software-design.html

[19] Jakob Engblom, "Transporting Bugs with Checkpoints," *Proceedings of the System, Software, SoC and Silicon Debug Conference (S4D 2010)*, Southampton, UK, September 15-16, 2010.