

# Virtual Platforms: Breaking New Grounds

Rainer Leupers

RWTH Aachen University, leupers@ice.rwth-aachen.de

Grant Martin

Tensilica Inc, gmartin@tensilica.com

Roman Plyaskin, Andreas Herkersdorf

TU Munich, herkersdorf@tum.de

Frank Schirrmeister

Cadence Design Systems Inc, franks@cadence.com

Tim Kogel

Synopsys Inc, tim.kogel@synopsys.com

Martin Vaupel

Robert Bosch GmbH, martin.vaupel@de.bosch.com

**Abstract**—The case for developing and using virtual platforms (VPs) has now been made. If developers of complex HW/SW systems are not using VPs for their current design, complexity of next generation designs demands for their adoption. In addition, the users of these complex systems are asking either for virtual or real platforms in order to develop and validate the software that runs on them, in context with the hardware that is used to deliver some of the functionality. Debugging the erroneous interactions of events and state in a modern platform when things go wrong is hard enough on a VP; on a real platform (such as an emulator or FPGA-based prototype) it can become impossible unless a new level of sophistication is offered. The priority now is to ensure that the capabilities of these platforms meet the requirements of every application domain for electronics and software-based product design. And to ensure that all the use cases are satisfied. A key requirement is to keep pace with Moore’s Law and the ever increasing embedded SW complexity by providing novel simulation technologies in every product release. This paper summarizes a special session focused on the latest applications and latest use cases for VPs. It gives an overview of where this technology is going and the impact on complex system design and verification.

## I. MORE REAL VALUE FOR VIRTUAL PLATFORMS

VPs are popular vehicles for various design tasks, such as early embedded SW development and HW platform architecture optimization. However, creating a VP from scratch for a new HW platform still means a huge investment of time, money, and manpower. Complex technologies have to be mastered, many platform components have to be modeled from scratch, and professional VP tools have a price tag, too. Enabling new applications and extending a typical VP’s lifetime would imply a higher return of investment. In turn, this requires R&D on various new technologies. We sketch three different routes here that, in our view, promise new opportunities for advanced VP use cases.

### 1. System-level power estimation

The design of new HW platforms, in particular in the telecom domain, is frequently driven by tight power consumption constraints. While VPs are already partially in use for post-silicon power optimizations, true electronic system

level (ESL) power estimation is getting more and more attention [1]. However, it is also known that power estimation without at least some gate-level and layout information is highly speculative at best. We therefore envision technologies as illustrated in Fig. 1, where abstract power models are first created and calibrated based on low level circuit information and afterwards are used stand-alone for accurate, yet very fast, power evaluation of different design points.

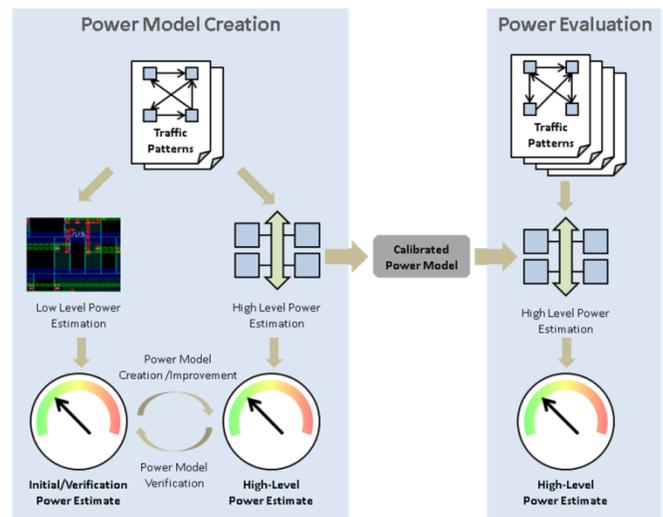


Fig. 1: High-level power model design methodology

This requires enhancements of current abstract processor simulators (e.g. [2]) as well as careful characterization of the key circuit parameters that determine the power estimation at higher levels.

### 2. Multicore SW verification

With the increasing number of processor cores inside SoC platforms, SW verification and debugging become pressing issues. We need to manage detection of new types of bugs in parallel SW applications as well as the complexity of debugging on dozens of cores simultaneously. VPs provide a great debug infrastructure due to maximum observability and

controllability. They should be extended towards systematically catching concurrent SW bugs. For instance, this can happen by means of an assertion mechanism that monitors SW event traces (e.g. shared memory accesses) and reports hazardous inter-process communication behavior.

### 3. Need for (simulation) speed

Simulation speed might well evolve as the key limiting factor for VP use. Multi/Manycore system complexity is still skyrocketing, and (quoting Lisa Su's DAC 2011 keynote) transistor counts are increasing faster than simulation tool speed. In the long term, this will prohibit some of the traditional VP use cases. Unfortunately, unlike in previous times, there is no universal turbocharger for ESL simulation in sight. We may thus expect that the need of keeping pace with Moore's Law will lead to VPs that adopt a blend of different approaches currently under investigation in research [3]: abstract/hybrid processor and NoC simulation, sampling based simulation, parallel and/or distributed simulation, just to name a few.

## II. CONFIGURABLE PROCESSOR-CENTRIC VIRTUAL PLATFORMS – NEW DOMAINS AND NEW USES

VPs and processor-centric design have gone together for many years. Configurable processors emphasize the need for virtual prototyping in both early design space exploration, where applications are partitioned and individual heterogeneous processors are configured with structural parameters and application-oriented instruction extensions, and in verification of back-end software development. Starting with proprietary C/C++ system modeling approaches, support for virtual prototyping has migrated to the use of SystemC as the modeling integration infrastructure, linking to C++ based instruction set simulation technology in both cycle-accurate and just-in-time host-compiled-code fast functional simulation forms. This base SystemC infrastructure has been integrated into a variety of commercial ESL virtual prototyping tools, which promotes interoperability with models of other IP and advanced analysis capabilities.

The applications for such virtual prototyping have grown in lock-step with the application of configurable, extensible processor technology. The domains of audio, video, and network processing were relatively early uses. More recently, wireless applications for both base stations and user equipment have pushed the envelope on heterogeneous multi-core simulations. Just the baseband subsystem for LTE can require a few to more than a dozen processors, ranging from highly application-specific engines to complex DSPs.

It is not possible to sit still with VPs. The growth in the number of cores in systems and the growth in the amount of software running on these cores demand frequent refreshes of technology. What was effective in fast-functional compiled-code simulation technology when first introduced – for example, running at 50 MIPS for simulating complex instruction extensions – becomes too slow when divided among ten processors rather than used for one. In addition, specific use-cases – such as mixes of peripherals with memory-mapped registers, some of which have side effects on writing (which is infrequent) and some of which are polled frequently,

may require finer granularity in direct memory access methods to achieve good performance. The heuristics used to control interpretation vs. compilation of hot code regions, the use of different compilers with different speed of compilation vs. speed of compiled code tradeoffs, and the use of multiple tasks and processors for compilation, all need upgrading as demand for performance increases, along with many other improvements for speed.

The need for more speed when using commercial ESL tools requires refreshes of integration and the capabilities supported in such integration. This can be an extensive many-month exercise, but given the importance of commercial virtual prototyping tools to many users, is again triggered by the increase in performance required for their platforms and use cases. This also triggers demand to support OSCI TLM-2.0 standards even when existing integrations support an equivalent set of transaction level modeling capabilities. The attractions of a standard for interoperability such as TLM-2.0, even though late in arriving, mean that more models will be built to this standard over time, increasing the desirability of supporting the TLM-2.0 ecosystem.

One extremely interesting use case for virtual prototyping technology with configurable processors could be called “virtual prototyping without SystemC Tears, virtual prototyping without commercial ESL \$\$\$”. This is a way of using configurable processor SystemC modeling technology to build subsystem VPs, with one processor or many, and drawing on a library of components such as memories, routers, arbiters, DMAs, and memory mapped peripherals. The technology has several intriguing characteristics:

- It does not require the modeler to know anything about SystemC or C++.
- It is interpreted, creating a SystemC subsystem model on the fly.
- It uses a very simple script-like notation to specify the subsystem.
- It uses inference to interconnect components.
- It can model a variety of interconnect structures.
- It can be easily integrated into an Eclipse based IDE to allow rapid subsystem model creation and software mapping to the heterogeneous set of processors.
- It can output a full static SystemC model of the subsystem, that can be developed further to incorporate IP from other sources and more complex interconnect models.

This technology has some constraints – it is limited to IP from one configurable processor vendor, and adding a new component to the interpreter requires a tedious set of 31 distinct steps. However, it is being evolved to add standard APIs that will greatly reduce the effort of adding new components to the library, thus extending its range of application further.

A final area of continued exploration and development is the evolution of ISS technology to allow simulations to run on multiple cores – “use multicore to design multicore” [4].

### III. ARCHITECTURE-LEVEL HARDWARE/SOFTWARE DESIGN SPACE EXPLORATION FOR MULTICORES

Extensive design space exploration requires scalable and accurate simulation methods and tools in order to keep up with the growing complexity of multiprocessor VPs. Conventional performance estimation methods, e.g. employing instruction set simulators (ISS), cannot satisfy the increasing requirements on the performance of ESL simulations. In order to speed up the exploration at the architecture and system level, it is essential to define a sufficient amount of simulation events and raise the abstraction level of the models. At the same time, higher levels of abstraction introduce new challenges in the accurate modeling of the components' timing behavior since micro-architectural details, such as instruction pipelining or out-of-order execution, are partially (or completely) abstracted.

Addressing the trade-off between performance and accuracy, trace-driven simulations accelerate the design space exploration by eliminating unnecessary details in the models [5, 6]. The idea behind this method is to capture and abstract the behavior of processing elements in the form of application traces. The collected trace data describe interactions between processing elements and are used to predict the system's behavior in different environments in a shorter time. The traces can be defined at different levels of detail depending on the points of interests in design space exploration. At the architecture level, the application traces capture the interaction between VP components while abstracting the internal events of the processing elements. The challenge is in generating a representative set of meaningful traces that cover realistic scenarios for the exploration. The proposed workflow of our approach consists of two steps shown in Fig. 2. In the first step, the application traces are obtained during a pre-evaluation phase by executing the target code on a reference cycle-accurate simulator which incorporates all necessary micro-architectural details. The simulator generates a trace consisting of bus accesses observed during the execution of the code and the time intervals between them.

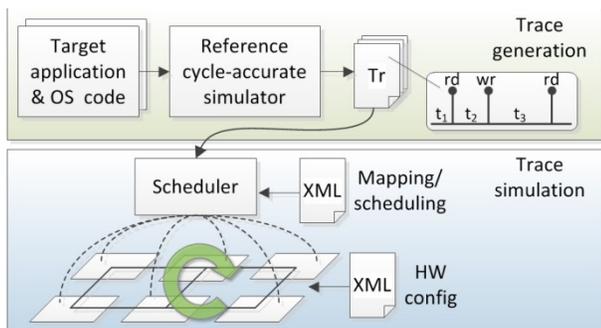


Fig. 2: Generation and simulation of abstracted traces for architecture-level design space exploration

Generated once, the application traces can be reused multiple times during the design space exploration. The traces are simulated at the architecture level in a trace-driven VP which contains abstracted models of the processing elements. These models reproduce the application workload captured in

the traces and stimulate the models of shared resources and interconnect. Due to the abstraction of internal processing, trace-driven simulation performs faster compared to full cycle-accurate model, achieving reported speedup factors of up to 174x [7]. The simulation performance is limited by the speed of system-level models of shared components (e.g. caches or on-chip interconnect) as well as by the performance of the simulation engine employed for the management of simulation events. In addition, the performance of trace-driven simulations is constrained by the IO-bandwidth of the hard disk which is used for storing and fetching the traces.

Along with the target applications, the traces can abstract the workload of the target operating system. By means of a high-level scheduler, the designer can perform more comprehensive design space exploration. The scheduler manages the execution of traces on the underlying core models, allowing for evaluation of various application mappings and scheduling policies at a higher level of abstraction [5].

The abstraction of functionality in the trace-based simulations requires a separate trace to be generated for each input applied to the target application. One way to tackle this problem is to reduce the granularity of traces. The trace can be sliced into smaller parts or *atomic traces*, according to the control flow graph of the target code (Fig. 3).

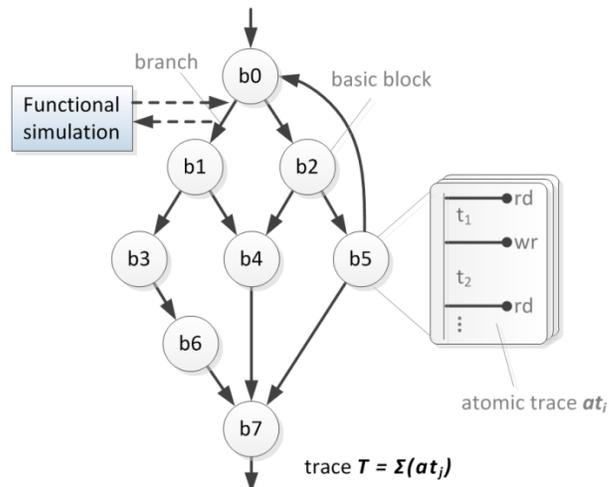


Fig. 3: Fine-granular atomic traces capturing the execution of basic blocks of the target code

The original trace can be reconstructed by concatenating the atomic traces according to the control flow of the target application, which can be changed at simulation run-time in this setup. The concatenation of atomic traces requires the values of target branch addresses which define the next traces to be executed. For this purpose, fast, untimed, functional simulation can be employed in order provide the control flow information (Fig. 3).

In advanced out-of-order processors, the timing of basic blocks is not static but strongly context-dependent. In order to address different execution patterns of the same basic block, more than one atomic trace per basic block has to be

considered. This approach requires run-time evaluation of the simulation context in order to determine which atomic trace of the block has to be executed [7]. The use of multiple atomic traces per basic block allows for accurate reconstruction of the processor's behavior at higher simulation speeds than in conventional ISS's.

#### IV. VIRTUAL PLATFORMS AND THEIR ECOSYSTEM

To understand VPs and their eco-system it is important to understand the design flow they are enabling. Fig. 4 shows an abstract system development flow starting from system modeling through hardware software partitioning leading into the three pillars of hardware development, software development and hardware/software integration. The hardware development flow re-uses as much IP as possible, generates new RTL from high-level Transaction Level Models (TLM) and then leads into the traditional EDA flows of SOC and Silicon Realization. The software flow equally re-uses as much IP as possible and implements from C and C++ using compilers the actual software stacks starting with firmware enabling operation systems through drivers, middleware and applications.

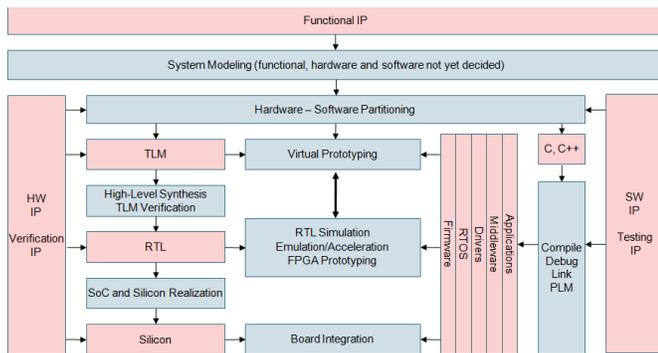


Fig. 4: An abstract system-development flow

For the center pillar of HW/SW integration, various techniques are in use today.

Virtual prototypes for software development are available pretty early in a project and are representing fully functional software models of systems on chip (SoCs), boards, I/Os and user interfaces. They execute unmodified production code, and they run close to real-time with virtualized external interfaces. They offer high system visibility and control, including multi-core debug. The speed of VPs will degrade to the single-digit MIPS range or even lower if users choose to mix in more timing-accurate models of the hardware, which is an approach often used for architectural analysis.

Variations of virtual platforms are so-called software development kits (SDKs) like the iPhone SDK. While SDKs offer most of the advantages of the standard virtual prototypes, their accuracy is often more limited because they may not represent the actual registers as accurately as virtual prototypes but instead allow programming toward higher-level application programming interfaces (APIs) and often require re-compilation of the code to the actual target processor after users have verified functionality on the host machine on which the SDK executes.

Prior to VPs and SDKs, system-level models may exist at a level of abstraction at which decisions about hardware and software have not been made. They typically use descriptions like UML or proprietary languages like MatLab.

Later areas of software and hardware integration use the register transfer level (RTL) used for hardware development. Because RTL takes time and effort to implement and integrate, design teams can be very hesitant to change the hardware architecture once RTL is available, unless major defects have been found.

After simulation confirms that the RTL is reasonably stable, emulation and acceleration provides a vehicle for hardware-assisted software development. It differs from FPGA prototypes in that it enables better automated mapping of RTL into the hardware together with faster compile times, but the execution speed will be lower and typically drop to the single-MIPS range or below.

Available later in the design flow, but still well before silicon, FPGA prototypes can serve as a vehicle for software development and integration as well. They are fully functional hardware representations of SoCs, boards and I/Os. They implement the ASIC RTL code and often run in the speed range of 10s of MIPS, with all external interfaces and stimulus connected. Due to the complexity and effort of mapping the RTL to traditional FPGA prototypes, it is not really feasible to use them before RTL verification has stabilized.

Finally, after the actual silicon is available, early prototype boards using first silicon samples can enable software development on the actual silicon. Once the chip is in production, very low-cost development boards can be made available. At this point, the prototype will run at real-time speed and full accuracy. Software debug is typically achieved with specific hardware connectors using the JTAG interface and connections to standard software debuggers.

At least ten different characteristics determine the applicability of the chosen prototyping approach and the models it is built from. They include time of availability in a project, execution speed, accuracy, bring-up cost, production cost, replication cost, debug insight into hardware, debug insight into software, execution control and availability of system interfaces to include the system environment for analysis and verification.

From Fig. 4 it becomes clear that various interfaces and connections can enable different use models for VPs. Architects for the hardware and software portions and their partitioning will appreciate executable representations of the system models which were used to make the initial design decisions. As a result, connections to high-level representations and hybrid execution can make sense for VPs to allow verification as well as performance analysis.

VPs for software development increasingly are becoming part of the verification flow for hardware. Low-level software often will be a required part of test benches for hardware verification, ensuring correctness of the HW/SW interface. But more and more also the higher level functions can become part of hardware verification flows. For that purpose, hybrids

of VPs with all RTL based integration techniques are becoming more popular. They allow users to optimize the balance between the advantages of the different techniques, specifically as it relates to the ten characteristics of prototyping approaches mentioned above.

### V. SOFTWARE-DRIVEN VERIFICATION

It is common knowledge that functional RTL verification consumes up to 70% of the total HW development time. Two major factors among that budget are the effort to create the testbench and the time for debugging of detected issues. Using SW running on VPs for functional verification adds value in both areas: First, VPs can be used as directed stimuli generators as well as reference models in RTL testbenches. Second, VPs can accelerate the debugging of detected issues. Despite the obvious advantages, so far only a few companies are successfully deploying SW Driven Verification (SDV). The slow adoption is mainly due to the high effort to establish this flow, which in turn is caused by the previous lack of standards in both the VP domain as well as the verification domain. Recently the standards in both areas are maturing and with that, a SDV flow can be established with little effort from standard building blocks.

The release of the TLM-2.0 standard in 2007 was inflection point for the broad adoption of VPs. Since then all IP vendors and EDA tool providers have migrated their proprietary model interfaces to TLM-2.0. By now, users can compose VPs from a broad range of Loosely Timed (LT) off-the-shelf components [8] and reap the benefits of VPs for early SW development and HW/SW integration.

On the verification side, the consolidation of vendor specific verification methodologies into the Universal Verification Methodology (UVM) fosters the broad availability of Verification IP (VIP) with standard SystemVerilog transaction-level interfaces. In addition, the modular structure of state-of-the-art testbenches enables the deployment of Loosely Timed SystemC models and VPs as reference models as well as for stimuli generation.

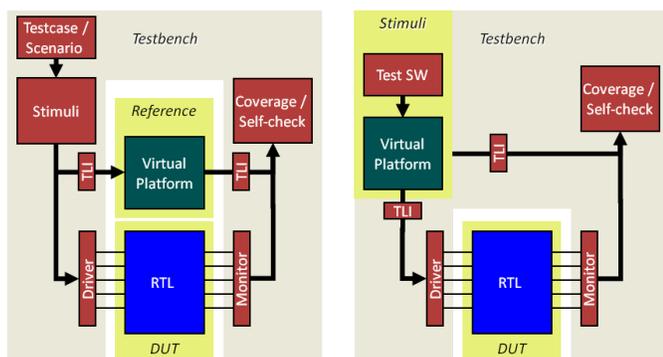


Fig. 5: Software Driven Verification Testbench

As depicted in Fig. 5, there are two main use models for SDV. The left side depicts the usage of a VP as a reference model in a SV testbench. By that, the creation of an additional SV reference model is not needed any more, which greatly reduces the testbench development effort. As an additional

benefit, the development and testing of the testbench itself can start early, long before the actual Device Under Test (DUT) becomes available. This enables the parallel creation of the DUT and the testbench and thus reduces the overall time to market.

The right side of Fig. 5 illustrates the usage of a VP as a stimuli generator for the DUT. As soon as an RTL block becomes available, it can replace its transaction-level model in the VP. Knowing that real Software and system scenarios are used greatly increases the confidence in the verification. Furthermore, the simulation speed is much faster, given that most of the system is simulated at the transaction-level.

The SDV use-models are based on the integration of VPs into SystemVerilog (SV) based verification environments. This setup relies heavily on existing testbench infrastructure like TLM-to-pin conversion with drivers and monitors as well as scoreboards for the smart comparison of the from the Loosely Timed reference and from the timed DUT. In addition the availability of the SystemC TLM-2.0 standard for VPs and the UVM verification standard enables the creation of highly reusable Transaction Level Interfaces (TLI) between the VP and the rest of the SystemVerilog testbench. Based on these off-the-shelf components it is very little effort to set up a SDV environment. As a result, we are starting to see a growing adoption of SDV.

### VI. VIRTUAL PLATFORMS FOR AUTOMOTIVE

While “hard” automotive applications have much in common with consumer electronics, the differences w.r.t. technical characteristics and constraints, development processes, and supply chains have to be taken into account before exploiting new methodologies in this domain, such as VPs. Hard automotive, in this context, means the focus on control-flow oriented, real-time, safety-critical applications such as engine control or electronic stability programs ESP™ - in contrast to, for example, infotainment systems.

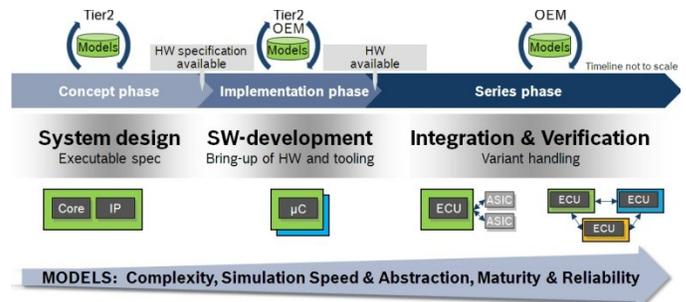


Fig. 6: Automotive ECU generation lifecycle

Similar to consumer devices, the user perceptible functionality of today’s automotive electronic control units (ECUs) is mainly defined by highly complex embedded software; and the challenges of this complexity make the use of VPs for system and software development attractive throughout the lifecycle of an ECU generation (Fig. 6): (1) In the System definition and partitioning phase models can aid communication between disciplines and companies and serve as executable specifications. (2) Before HW is available (“pre-silicon”) development can be accelerated by parallelizing the SW-development, adapting the SW tooling (e.g. debugger and

compiler), and preparing HW bring-up. (3) During the lifetime of a generation (“post-silicon”), where the advantages of virtual over real HW such as scalability, improved distribution, or automation and replication capabilities can be exploited. While better observability and controllability are properties of VPs for all kinds of applications (notably involving Multicore  $\mu$ Cs), in the context of automotive safety requirements (e.g. as stated in ISO26262 [9]) they seem particularly appealing for code coverage measurements and fault injection. In contrast to the consumer electronics business, this phase spans several years during which the tool environment has to be stable. Many variants are developed and maintained which have to be supported by the VP environment by means of a robust variant and version handling mechanism as well as by a scriptable model assembly methodology.

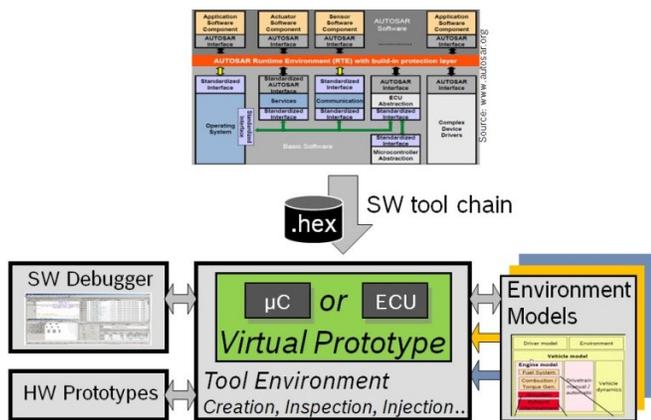


Fig. 7: Virtual Prototypes in the automotive context

The above use cases in combination with the involved (AUTOSAR [10])-SW layers (see Fig. 7) inflict a tremendous diversity of requirements on the used VPs w.r.t. complexity (IP blocks,  $\mu$ Cs, ECUs, networks of ECUs), accuracy and simulation speed (i.e. abstraction level), and maturity, which should be handled optimally by the same tool environment. On the other hand, the automotive supply chain becomes more diverse and fragmented, meaning that the models used by the different players must be able to interact seamlessly regardless of which tool was used to create them and which is used to integrate and execute them. In addition, these cyber-physical systems interact heavily and in hard real-time with their physical environment by means of a multitude of directly attached sensors and actuators in addition to communicating to other ECUs over networks. As a consequence, purely electronic (digital and analog) VPs are not sufficient: they have to interact seamlessly with existing simulation environments specializing in e.g. hydraulic, mechanical, electrical, and behavioral properties. The same holds for the sophisticated

legacy SW-tool chains and existing development processes, where the new methodology has to fit into in order to preserve the considerable investments made.

VPs will find wide-spread use in automotive, only if (1) the above issues are addressed and (2) the cost of modeling is bearable, which can only be accomplished by establishing open automotive modeling standards that guarantee the interoperability of models and tools within the whole ecosystem.

## VIII. CONCLUSIONS

We have summarized the state-of-the-art, current trends, and future needs in the domain of virtual prototyping. VPs remain an extremely interesting and fast developing topic, in terms of both academic research and new business directions. VPs are also an excellent example of an EDA domain, where academia and industry have shown a very fruitful collaboration in the past two decades. In order to continue this success story, the research community should address the fundamental technical issues, such as speed, abstraction, parallelization, reuse, and interoperability. At the same time, advanced use cases (including system optimization and verification) and new applications domains (such as automotive) offer fresh perspectives and provide ESL vendors with new marketing opportunities.

## REFERENCES

- [1] C.W. Hsu, J.L. Liao, S.C. Fang et al.: *PowerDepot: Integrating IP Based Power Modeling with ESL Power Analysis for Multi-Core SoC Designs*, DAC, 2011
- [2] T. Kempf, M. Doerper, T. Kogel et al.: *A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multiprocessor SoC Platforms*, DATE, 2005
- [3] R. Leupers, O. Temam (eds.): *Processor and System-on-Chip Simulation*, Springer, 2010
- [4] R. Leupers, L. Eeckhout, G. Martin, F. Schirmermeister, N. Topham, X. Chen: *Virtual Manycore platforms: Moving towards 100+ processor cores*, DATE, 2011
- [5] R. Plyaskin, A. Masrur, M. Geier, S. Chakraborty, A. Herkersdorf: *High-level timing analysis of concurrent applications on MPSoC platforms using memory-aware trace-driven simulation*, 18th IEEE/IFIP International Conference on VLSI and System-on-Chip (VLSI-SoC), Madrid, Spain, September 27-29, 2010
- [6] H. Lee, L. Jin, K. Lee, S. Demetriades, M. Moeng, and S. Cho: *Two-phase trace-driven simulation (TPTS): a fast multicore processor architecture simulation approach*, *Softw. Pract. Exper.*, vol. 40, no. 3, pp. 239-258, 2010.
- [7] R. Plyaskin, A. Herkersdorf: *Context-aware compiled simulation of out-of-order processor behavior based on atomic traces*, 19th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Hong Kong, October 3-5, 2011
- [8] www.tlmcentral.com
- [9] www.iso.org
- [10] www.autosar.org