Synopsys Virtual Prototyping for Software Development and Early Architecture Analysis

34

Tim Kogel

Abstract

This chapter summarizes more than 20 years of experience by the virtual prototyping group of Synopsys in the commercial deployment of Hardware/Software Codesign (HSCD). The goal of HSCD has always been to reduce time to market, increase design productivity, and improve the quality of results. From all the different facets of HSCD, virtual prototyping – complemented by links to emulation and FPGA prototyping – has so far proven to achieve the best return of investment with respect to these goals. This chapter first gives an overview of the main virtual prototyping use cases in the context of an end-to-end prototyping flow, which also includes physical prototyping and hybrid prototyping. The second part introduces the SystemC Transaction-Level Model (TLM) standard and the Unified Power Format (UPF) as the main modeling languages for the creation of Virtual Prototypes (VPs) and system-level power models. The main body of this chapter focuses on the commercially deployed virtual prototyping use cases for architecture exploration and system-level power analysis.

Acronyms

API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AT	Approximately Timed
AT-BP	Approximately Timed Base Protocol
AV	Architects View
AXI	Advanced eXtensible Interface
CA	Cycle Accurate
CPU	Central Processing Unit
DDR	Double Data Rate
DMA	Direct Memory Access

T. Kogel (✉)
Synopsys, Inc., Aachen, Germany
e-mail: tim.kogel@synopsys.com

S. Ha, J. Teich (eds.), *Handbook of Hardware/Software Codesign*,
DOI 10.1007/978-94-017-7267-9_34

DMI	Direct Memory Interface
DRAM	Dynamic Random-Access Memory
DSP	Digital Signal Processor
DVFS	Dynamic Voltage and Frequency Scaling
ECU	Electronic Control Unit
FPGA	Field-Programmable Gate Array
FT	Fast Timed
GFRBM	Generic File Reader Bus Master
GPU	Graphics Processing Unit
HAPS	High-performance ASIC Prototyping System
HDL	Hardware Description Language
HSCD	Hardware/Software Codesign
HW	Hardware
IP	Intellectual Property
ISS	Instruction-Set Simulator
LT	Loosely Timed
MCO	Multi-Core Optimization
MPSoC	Multi-Processor System-on-Chip
OS	Operating System
PMU	Power Management Unit
QoS	Quality of Service
RFTS	Run Fast Then Stop
RTL	Register Transfer Level
SCML	SystemC Modeling Library
SLP	System-Level Power
SMP	Symmetric Multi-Processing
SoC	System-on-Chip
SW	Software
TCL	Tool Command Language
TLM	Transaction-Level Model
UPF	Unified Power Format
VPU	Virtual Processing Unit
VP	Virtual Prototype

Contents

34.1 Introduction

In a traditional development process, the hardware/software integration and validation can only start after the hardware development is finished and the first silicon samples are available. This forces a sequential dependency between the Hardware (HW) and Software (SW) development phases and puts a lot of stress on the overall schedule. Prototyping offers a set of methodologies to overcome this dependency by "shifting left" the architecture design and software development flows.

Today, the following variants of prototypes are in deployment by semiconductor and electronic system companies:

- **Virtual prototypes** are fast, executable models of the System-on-Chip (SoC). They are typically created from SystemC Transaction-Level Models (TLMs) that are delivered by semiconductor Intellectual Property (IP) companies and are extended with models that are specifically created for the SoC in development (see Sect. 34.2.1). Different types of Virtual Prototypes (VPs) are created based on the requirements for simulation speed and timing accuracy: Synopsys offers Platform Architect for Multi-Core Optimization (MCO) for the creation and usage of VPs for architecture design as well as Virtualizer™ for software development related use cases.
- **Physical prototypes** provide specialized FPGA-based systems and tools to execute Register Transfer Level (RTL) implementations at high speeds. This way, Field-Programmable Gate Array (FPGA) prototypes are useful for system validation and software development purposes. Synopsys provides the HAPS® high-performance Application-Specific Integrated Circuit (ASIC) prototyping system.
- **Hybrid prototypes** combine a VP with a physical prototype; see also Sect. 3 in the ▸ Chap. 37, "Control/Architecture Codesign for Cyber-Physical Systems". For target use cases like IP driver development, hybrid prototypes offer users a way to optimize the prototyping setup based on the availability of TLMs and RTL implementations. Taken together, Synopsys Virtualizer™ and HAPS® provide a hybrid prototyping environment.

All three types of prototypes typically require a dedicated team inside a semiconductor organization that specializes in providing these prototypes to the actual end users.

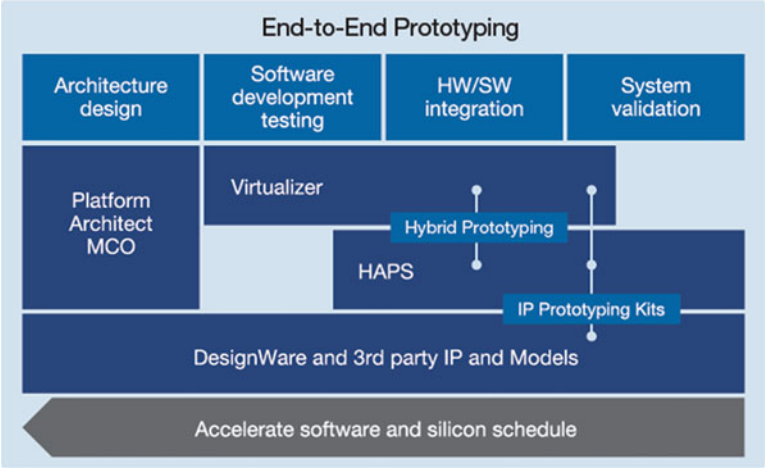

Fig. 34.1 Design tasks and solutions in end-to-end prototyping

As depicted in Fig. 34.1, these prototyping methods can be applied for multiple tasks in a software-driven SoC design flow: architecture design, software development and testing, HW/SW integration, and system validation. The biggest value is achieved when they are applied across all the stages of SoC design.

The following paragraphs review these tasks individually.

34.1.1 Architecture Design

Architect teams are typically working in several stages and have projects that deal with generation N+2, where N refers to the current SoC generation in production. Traditionally, architects rely on past experience and static spreadsheet analysis for estimating power and performance of the next- or second-next-generation product. This manual and static analysis is becoming increasingly difficult due to the increasing complexity of electronic products. Virtual prototyping enables architects to simulate the impact of critical application use cases and of specific design decisions on the overall performance and power consumption; see also the paragraph on architectural virtual platforms in Sect. 2 of ▶ Chap. 33, "Hardware/Software Codesign Across Many Cadence Technologies".

To benefit from early architecture exploration using a VP, an architect needs three fundamental ingredients: (1) performance models, (2) power models, and (3) application scenarios.

Performance models describe relevant components of the SoC, such as the interconnect and memory subsystems, with sufficient accuracy to enable critical design decisions based on the latency and throughput data provided by simulating these performance models.

Power models are needed to also analyze the expected power and energy consumption for the main components that consume power on the SoC. These power

models rely on power modeling standards [11] augmented with power consumption data from IP data sheets and with measurements during the implementation steps from previous projects.

Finally, the processing and communication requirements of the software that will actually run on the chip needs to be described at an abstract level in terms of an application workload model. Virtual prototyping offers ways to either manually capture these task-based workload models or to extract them from software execution traces.

The result is that the performance and power of the architecture can be explored more accurately compared to relying on static spreadsheet-based exploration. Thanks to their flexibility and high simulation speed, VPs enable the simulation of orders of magnitude more architecture variations compared to doing the same at the RTL. This typically leads to double-digit gains in terms of power and performance trade-offs, reducing the cost and excessive power consumption of over-designed products. Section 34.3 elaborates more on the different aspects of virtual prototyping for architecture analysis and optimization.

34.1.2 Software Development and Testing

The software development schedule can achieve the biggest time to market gains by applying prototyping. Starting software development earlier and, thus, shortening the overall schedule as well as enabling earlier feedback between hardware and software teams is a real game changer.

For many years, semiconductor companies have been using only physical prototyping to do software development, typically for IP- and subsystem-related software. Most of them have adopted commercial FPGA-based prototyping solutions to benefit from existing design and debug automation tools to be able to achieve the fastest time to first prototype and optimize for highest performance (see also Sect. 4 in ▸ Chap. 33, "Hardware/Software Codesign Across Many Cadence Technologies"). The market is now shifting rapidly to using integrated commercial solutions, comprised of hardware and software tools to achieve a high-performance prototype in weeks rather than months, hence, significantly increasing the useful time of prototyping before silicon arrival.

With the advent of larger FPGAs, these physical prototyping systems are usable to prototype much larger portions of the SoC, including Graphics Processing Units (GPUs), enabling more software development early in the design cycle. While FPGA-based prototyping has proven to provide high value by enabling early software development, virtual prototyping has been adopted by many semiconductor vendors as a complimentary prototyping solution. By creating and deploying VPs, semiconductor vendors are able to further shift left their software development and start more than 12 months before silicon availability.

Where FPGA-based prototyping offers key benefits enabling HW/SW integration and especially system validation, virtual prototyping provides key capabilities to accelerate software development and scale software testing. Since VPs are based on models, they are not dependent on RTL implementation availability, and, hence,

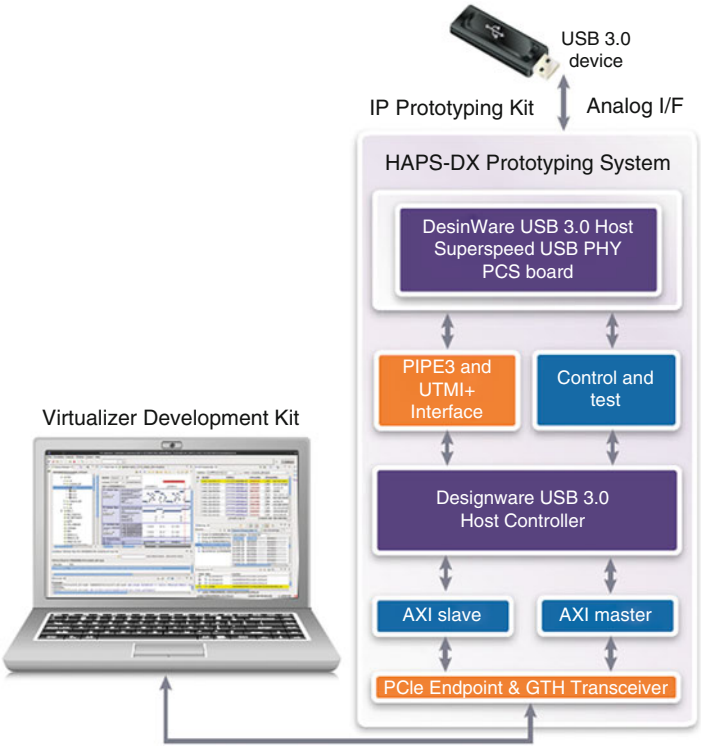

Fig. 34.2 Example DesignWare hybrid IP prototyping kit

they enable software development much earlier in the design cycle. Another benefit of using models rather than RTL execution on a physical prototype is the improved control and debug visibility provided by VPs. They also allow for fault injection to test how the system will react [21], and they scale more easily to massive parallel testing and, thus, help increase software quality.

As shown in Fig. 34.2, virtual and physical prototyping environments can be merged into a hybrid prototyping environment, which offers a great solution for IP-specific software bring up. Early availability of models and thus VPs for new processors combined with IP mapped on an FPGA offer an early and functionally complete solution for IP-specific software development, testing, and HW/SW integration.

34.1.3 Hardware/Software Integration and System Validation

Hardware/software integration typically starts at the subsystem level on an FPGA-based prototype, when RTL IP blocks or subsystems that are considered to be relatively stable are integrated with firmware or drivers. The bring-up time can

be significantly reduced when VPs or hybrid prototypes are used for the software development so that the software is more or less functionally complete by the time of the HW/SW integration step.

System validation deals with removing the uncertainty of how the SoC with the software stack will perform in the target environment. The physical prototype is plugged into the real-world environment and performs realistic scenarios. Examples of this can be system validation of a networking SoC inside an actual networking device for which execution speeds of multiple 10s of MHz are required. Or validation of application processor SoCs that have to support many different interface protocols such as USB, eMMC, and others.

34.1.4 System-Level Power Analysis

Energy proportional computing is a key concern for many electronic products, most notably for any battery-driven mobile consumer device [6]. Any device should consume only as much power as absolutely required to perform a certain task. The increasing number of Central Processing Units (CPUs) with their high frequencies, the increasing size of LCD screens and cameras, and the multitude of radios and sensors are driving the total power consumption beyond what is acceptable by the consumer. Only proper management of the power states within these devices allows minimizing the total power consumption.

Both architecture and software have a significant impact on power consumption:

- The key decisions impacting power consumption are taken during the architecture definition phase. For example, a relevant use case for a smartphone could be the streaming of a video via the cellular network and displaying it in HD resolution on a connected LCD screen. Each of the use cases requires the services of a certain set of components. This use case analysis drives the partitioning of the device into power domains and their respective operating points. Typically, the supply voltage and frequency of the power domains can be controlled individually to provide the flexibility to later optimize the power dissipation and energy consumption of the different use cases. The optimal definition of power domains and operating points is key to achieve the goal of an energy proportional system.
- Software plays a significant role in the device power management at run time. The software controls and drives hardware components which actually consume power. Software stacks such as Android/Linux with millions of lines of code implement various power saving strategies on almost each software layer starting at the driver and ending up in the application layer [7]. A software power inefficiency or malfunction can quickly cause a 5× drop in standby time.

Power consumption is an orthogonal aspect, which caters to all prototyping use cases. Therefore the modeling of the power consumption should be as much as

possible independent of the actual prototyping itself. Section 34.2.3 shows how to add component-level power models as an overlay to a VP based on the IEEE 1801-2015 UPF-3.0 standard [11]. Using such power models enables architects as well as software developers to take the impact of their design decisions and software implementation on the power consumption into account.

34.1.5 Summary

By leveraging the different technologies in the context of an end-to-end prototyping solution, the typical design time line is significantly reduced.

The shift left impact of end-to-end prototyping not only reduces the overall design time line and hence the time-to-market but also helps companies that are deploying this methodology validate that their products better match the original design requirements. By optimizing the architecture early on in context of the software scenarios and designing the hardware and software side by side, the resulting product is better balanced.

By now, virtual prototyping for early software development is already a widely deployed methodology [4, 23]. Refer also to Sect. 2 in ▸ Chap. 33, "Hardware/Software Codesign Across Many Cadence Technologies". Therefore the main body of this chapter focuses more on the virtual prototyping use cases for architecture exploration and system-level power analysis. Before going there, the next section first introduces the modeling methodologies, which are the foundation for creating VPs.

34.2 Modeling for Virtual Prototyping

Modeling is the key initial task for creating a VP. This task requires the specification of the system or SoC to be modeled, the modeling tools, and knowledge of modeling languages. This section first focuses on SystemC TLMs, which are the established lingua franca for the creation of VPs. The second part gives an introduction to UPF-3.0, the new modeling standard for system-level power analysis.

34.2.1 The SystemC Transaction-Level Modeling Standard

The IEEE 1666 standard for SystemC and TLM defines the widely accepted modeling language for the creation of VPs [12]. SystemC is a C++ library providing a set of classes to model system components and their communication interfaces, plus a cooperative multitasking environment to model concurrent activity in a system. On top of SystemC, the TLM library supports a modeling style where the communication interfaces between system components is not based on individual signals, but on a set of function calls and a payload representing the full semantics of the communication interface. This reduces the number of synchronization points

between communicating component models, which in turn greatly improves the overall speed of the event-driven SystemC simulation kernel. Since 2008, the TLM-2.0 standard provides a well-defined set of Application Programming Interfaces (APIs) and payload constructs to create interoperable TLMs for memory-map-based communication protocols.

The IEEE Std 1666 TLM-2.0 Language Reference Manual [12] identifies the following coding styles:

- The **Loosely Timed (LT)** modeling style aims to maximize the simulation speed of a model by abstracting the communication to the highest level and by minimizing the synchronization overhead.
- The **Approximately Timed (AT)** modeling style focuses on the timing of the transactions between different components in a system by providing multiple timing points for each transaction.

As illustrated in Fig. 34.3, both LT and AT modeling styles use the same concept of sockets, generic payload, and an extension mechanism for modeling memory-mapped communication protocols. The extension mechanism allows adding of custom attributes to the generic payload, which is important to model protocol-specific attributes, like the transaction id in the Advanced eXtensible Interface (AXI) protocol [1]. This common infrastructure enables the smooth integration of models using different modeling styles. On the other hand, LT and AT leverage specific mechanisms to cater to the specific requirements of different virtual prototyping use cases for software development and architecture analysis.

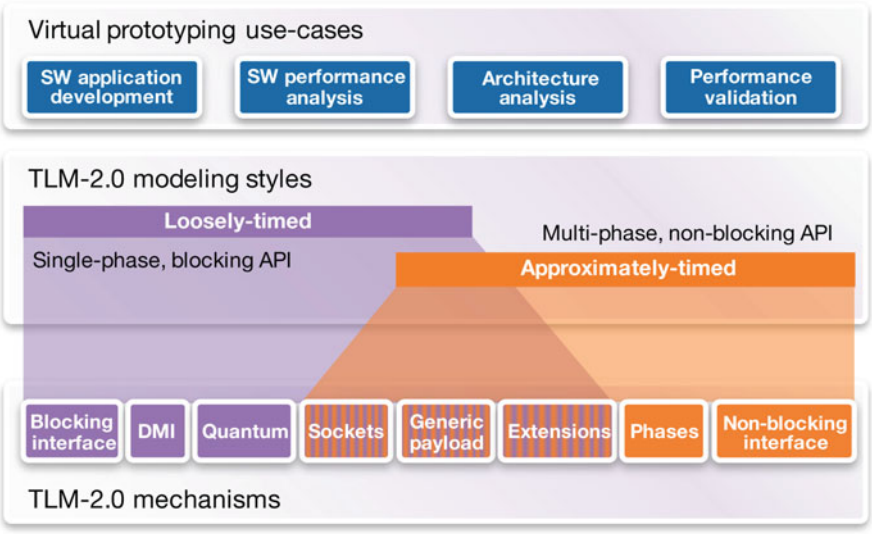

Fig. 34.3 TLM-2.0 modeling styles and mechanisms

34.2.1.1 Loosely Timed Modeling Style

VPs for software development are created in order to provide an abstract model of the target hardware platform, which can execute the unmodified software. The key requirements are:

- Simulation speed: It is important that the VP can execute software at a speed that is as close as possible to real time of the actual target device.
- Register accuracy: In order to run embedded software correctly, the memory and memory-mapped register layout and content should be modeled.
- Functional fidelity: All relevant responses of the target hardware should be modeled.

The LT modeling style is intended to maximize the execution speed while providing the minimal level of timing fidelity. The key concepts in the TLM-2.0 standard to achieve high simulation speed on top of the event-driven SystemC simulation kernel are temporal decoupling, the Direct Memory Interface (DMI), and blocking communication:

- Temporal decoupling allows initiator components, like processor models, to run ahead of the global time for a maximum quantum of time before synchronizing with the SystemC kernel; see also Sect. 1.3 in ▸ Chap. 19, "Host-Compiled Simulation".
- DMI allows initiator components to bypass the regular TLM interface and directly access instruction and data memory via the simulation host address.
- For non-DMI access to memories and peripheral registers, the simple blocking TLM transport interface is used. As depicted on the left side of Fig. 34.4, LT communication is modeled using a single function call.

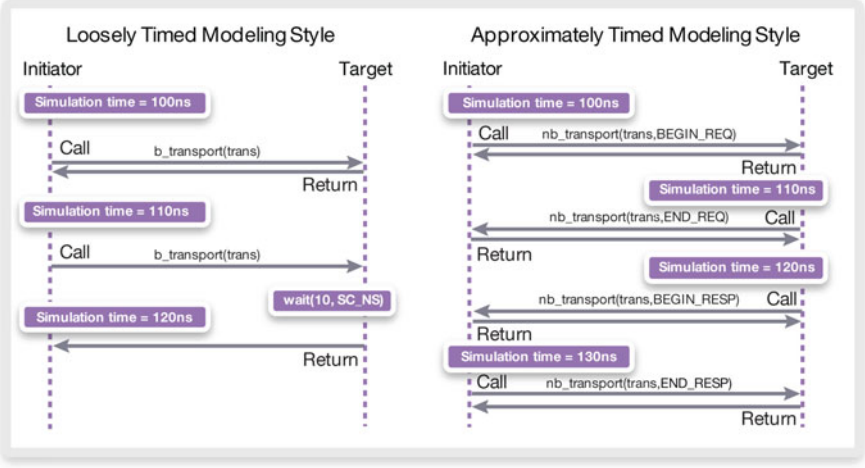

Fig. 34.4 TLM-2.0 Loosely Timed (*left*) and Approximately Timed (*right*) protocols

The LT modeling style reflects the hardware interactions with software and how register content is updated. For example, timer interrupts happen roughly at the intended time to simulate the timing calibration loop in a Linux boot or to execute real-time software in automotive Electronic Control Units (ECUs).

34.2.1.2 Extended Loosely Timed Modeling Style

The TLM-2.0 generic payload only covers the common subset of transaction attributes like address, data, and burst length. The extension mechanism allows to include additional protocol-specific attributes to the generic payload, e.g., security extensions, atomic transactions, coherency flags, etc. Based on this extension mechanism, owners of on-chip bus protocols have defined a layer on top of TLM-2.0 for creating protocol-specific models in a interoperable way [2].

The loosely timed modeling style has been very successful in fostering the availability of interoperable models from all major IP providers [3, 25]. The availability of LT TLMs for off-the-shelf IP blocks has significantly reduced the investment for creating VPs for software development.

34.2.1.3 Approximately Timed Modeling Style

VPs for early architecture analysis and exploration are created in order to provide an abstract model of the target hardware, which reflects relevant performance metrics, e.g., bandwidth, throughput, utilization, and contention. The key requirements are:

- Scalable timing accuracy: The accuracy requirements depend on the goal of the project. For example, an abstract model of a DRAM is good enough for exploring HW/SW partitioning, but a highly accurate model is needed for optimizing the configuration of the DRAM memory controller.
- Compositional timing: The end-to-end performance of a system can be obtained from assembling a set of components which only model their individual timing.

Compared to the LT modeling style described previously, the AT modeling style is intended to model the communication with more detailed timing. As shown on the right side of Fig. 34.4, a single transaction is broken into multiple phases to reflect the timing of a bus protocol in more detail. The non-blocking TLM transport interface is used to mark start and end of each phase.

The TLM-2.0 initiator and target sockets bundle a forward and backward path in one interface to enable bi-directional communication. The initiators calls nb_transport to mark the begin of a request phase and sometime later the target calls nb_transport to mark the end of a request phase. As depicted in Fig. 34.4, the TLM-2.0 standard defines an Approximately Timed Base Protocol (AT-BP) with a request and response phase marked by four distinct timing points. This enables the modeling of basic communication aspects like throughput, latency, and transaction pipelining.

34.2.1.4 Extended Approximately Timed

The TLM-2.0 AT-BP has limited expressiveness when it comes to accurately representing any real-life on-chip bus protocols:

- It does not provide with timing points for the individual data beats of a burst transfer. This becomes particularly problematic when interfacing TLM-2.0 AT-BP with Cycle Accurate (CA) or RTL models.
- It requires all address and data information to be available for writes at the start of the transaction.
- It is not possible to have concurrent read and write requests, as required by, e.g., the AMBA AXI protocol [1].

To overcome these deficiencies of the AT-BP, the TLM-2.0 standard provides an extension mechanism for the AT modeling style, which enables the definition of additional protocol phases and timing points. Together with the extension mechanism for the generic payload, which is also used for loosely timed modeling, this enables the more accurate modeling of on-chip bus protocols. In fact, the AT extension mechanism allows the definition of fully CA modeling of real-life bus protocols.

The issue is that protocol-specific extensions break the interoperability between AT-BP and extended AT models. Synopsys has defined a Fast Timed (FT) modeling infrastructure. FT is based on the TLM-2.0 AT mechanism and enables the definition of more accurate protocols while preserving interoperability with the AT-BP:

- Each protocol extends the generic payload with an attribute indicating the current state in the protocol state machine.
- For each protocol, protocol-specific attributes are added as needed, e.g., for cacheability, out-of-order transactions, etc. This should be limited to those attributes that are not already covered by the TLM-2.0 AT-BP. These extensions are ignorable in the sense that a model should assume they have a default value in case they are not present in the payload.

The idea is that FT protocols remain compatible with the TLM-2.0 AT-BP and rely on extended sockets and payload to provide the necessary protocol conversion logic so that conversions are only done when required and can be inserted automatically.

34.2.1.5 Summary

The goal of the IEEE 1666 TLM-2.0 standard is to enable model interoperability at the level of SoC building blocks, e.g., processors, buses, memories, peripherals. For this purpose, TLM-2.0 standardizes the modeling interface for memory-mapped bus communication, which is the prevalent SoC interconnect mechanism. The LT and AT modeling styles cater to the different requirements of different use cases like software development and architecture analysis. Although AT allows more detailed timing modeling than LT, the modeling style of the communication interface should

not be confused with the abstraction level or timing accuracy of a model itself. LT and AT only refer to the communication aspect, whereas abstraction and timing accuracy also depend on the timing and granularity of the structure and behavior inside the component.

34.2.2 Modeling Objects and Patterns

Thanks to the TLM-2.0 interoperability standard, today many TLM-2.0 compliant models of standard SoC components like processors, buses, and memories are available from the respective IP provider. However, there are still a significant number of custom building blocks, e.g., timers, interrupt controllers, Direct Memory Access (DMA) controllers, or HW accelerators, for which specific models need to be created. TLM-2.0 defines the interoperability standard, but it does not prescribe how to model the internal behavior. In order to reduce the actual modeling effort, a well-defined modeling methodology and a library of reusable modeling objects is required. In larger companies, this is especially important to unify the modeling style across distributed modeling teams.

This section explains the concept of modeling objects and patterns based on the publicly available SystemC Modeling Library (SCML) from Synopsys [27].

34.2.2.1 The SystemC Modeling Library (SCML)

SCML is a layer on top of SystemC and TLM-2.0. It hides a lot of the complexity and common code that is required to correctly manage TLM-2.0 transactions, and it provides with modeling objects that handle common aspects of VP modeling. The modeling objects in the SCML promote the separating communication, behavior, and timing [14]. This way, the models created based on this methodology support different modeling styles like LT, AT, and FT.

Figure 34.5 illustrates the coding style, which is enabled by the SCML modeling objects:

- The interface to the interconnect model is separated from the actual behavior of the component. For the behavior of the component, a generic TLM-2.0 compliant LT, AT, or FT bus interface can be used.
- The interface between the extended protocol and the generic TLM-2.0 protocol used by the SCML storage objects is implemented by a protocol adaptation layer.
- The actual behavior of the component can be separated into a storage and synchronization layer and the pure functional behavior of the model.
  - The storage and synchronization layer stores the data of write transactions and returns the data in case of read transactions.
  - The behavior models the algorithm or state machine of the component. The behavior is triggered when certain memories or registers in the storage layer are accessed.

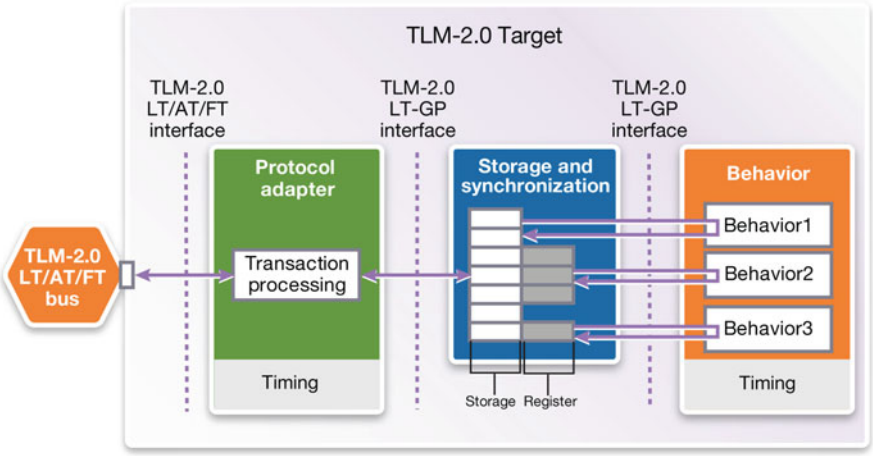

Fig. 34.5 SCML-based modeling pattern for target peripherals

- Finally, the different needs in timing accuracy can be addressed by separating the code that models the timing of the component from the pure functional behavior. SCML supports this separation by providing modeling objects for each of these layers.
  - The adaptation layer handles communication-related and data-independent timing aspects, e.g., the duration of a protocol phase or the number of outstanding transactions.
  - The behavior layer handles processing-related and data-dependent timing aspects.

The SCML modeling library greatly helps to reduce the modeling effort. In the context of commercial virtual prototyping projects, the effort for creating models can be further reduced by using model generation tools. For example, an SCML-based peripheral model can be automatically generated from an IP-XACT description of the register interface. Note that for the purpose of generating the register interface of a peripheral model, the IP-XACT importer only takes a subset of the IP-XACT standard into account, which is related to meta-data, register, and parameter information and which are supported by the SCML modeling objects.

34.2.3 System-Level Power Analysis

So far, the focus of this section has been on modeling functionality and timing, where the requirements are quite different depending on whether the VP is used for early architecture analysis or for software development. This section shows how to enable early power analysis irrespective of virtual prototyping use case.

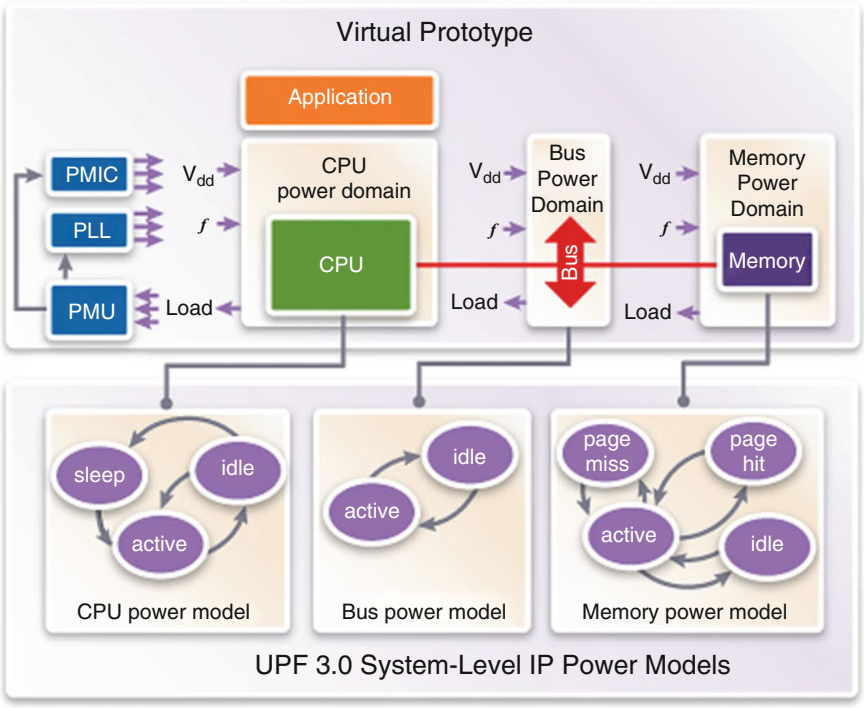

Fig. 34.6 Adding system-level power model as an overlay to VPs

The annotation of power information is performed through the use of system-level IP power models, which are power models of IP components specifically for use in system-level design. The format of system-level IP power models is defined by the IEEE 1801-2015 standard [11]. Originally the IEEE 1801 Unified Power Format (UPF) was defined to capture power intent for hardware implementation and verification. UPF is a format based on the Tool Command Language (TCL) [28] and defines the power supply and low power details as an overlay to the actual Hardware Description Language (HDL) implementation [10]. The new System-Level Power (SLP) features of the 1801-2015 "UPF-3.0" release extend the UPF TCL syntax to model power consumption as an overlay to a "host." Figure 34.6 shows an example of a VP with a UPF-3.0 system-level power overlay model. In this context, the UPF-3.0 power model calculates power consumption by observing the dynamic activity in the VP.

34.2.3.1 UPF-3.0 System-Level IP Power Models

A context-independent UPF-3.0 system-level power model comprises the following aspects:

- A set of buildtime and run-time parameters, which influence the power consumption of the respective IP. Examples of buildtime parameters are technology, size of a memory, or number of CPU cores. Examples of run-time parameters are voltage, frequency, or temperature.
- The set of relevant power states of the IP. These are not necessarily identical with the power supply states (e.g., `off`, `sleep`, `active`) but can also refer to operating modes with distinct power consumption signatures, e.g., a CPU in `WFI` (Wait For Interrupt) state or a video IP in `encode` or `decode` state.
- A set of functions to calculate the power consumption of the IP in the respective state and based on the set of parameters. The power functions need to separately return static and dynamic power consumption.
- The set of legal and illegal state transitions.

Now that the IEEE standard is ratified, it is expected that system-level IP power models are developed and distributed by IP teams (whether they be IP vendors or IP implementation teams within larger platform development groups).

34.2.3.2 UPF-3.0 System-Level Power Example

The example of a simple CPU power model in Fig. 34.7 illustrates the basic concepts of a UPF-3.0 power model:

- The power model is defined inside the `begin_power_model` and `end_power_model` commands.
- The `create_power_domain` command allows to represent hierarchy inside the power model, e.g., to represent different power states for the core and the cache inside one power model.

```
begin_power_model CPU
create_power_domain CORE -elements {}

add_parameter vdd                       -type runtime -default 1V          -description "voltage"
add_parameter frequency                 -type runtime -default 1000MHz     -description "frequency"
add_parameter temperature               -type runtime -default 25.0C       -description "temperature"
add_parameter percent_active_power_in_wfi    -type buildtime -default 10   -description ""
add_parameter mA_per_MHz_at_1V               -type buildtime -default 0.3  -description ""
add_parameter I_lkg_in_mA_at_1V_at_25C       -type buildtime -default 10mA -description ""

add_power_state -domain CORE \
    -state { OFF -power_expr {0mW 0mW} } \
    -state { WFI -power_expr { wfi_power {vdd frequency temperature} } } \
    -state { ACTIVE -power_expr {active_power {vdd frequency temperature} } }

add_state_transition -domain CORE \
     -transition {to_wfi      -from {ACTIVE OFF    -to WFI             -legal} \
     -transition {from_wfi    -from {WFI           -to {ACTIVE OFF}    -legal} \
     -transition {off         -from WFI            -to OFF             -legal} \
     -transition {on          -from OFF            -to WFI             -legal} \
     -transition {illegal1    -from OFF            -to ACTIVE          -illegal} \
     -transition {illegal2    -from ACTIVE         -to OFF             -illegal}
end_power_model
```

Fig. 34.7 Example of context-independent UPF-3.0 system-level power model

- The `add_parameter` command in the CPU power model defines three static buildtime parameters and three dynamic run-time parameters.
- The `add_power_state` command in the CPU power model defines three power states. The power consumption for the state `OFF` state is zero. For the other two states, the power consumption is calculated by the respective `wfi_power` and `active_power` functions. Both functions are sensitive to all of the three run-time parameters. This implies that the power functions need to be reevaluated whenever any of the run-time parameters in the sensitivity list changes.
- The `add_state_transition` command defines the set of legal and illegal state transitions.

The actual power functions are implemented outside of the scope of the actual power model and therefore not shown in this example.

In a second step, this context-independent power model can be instantiated in the context of a VP, which contains a TLM of a CPU. The corresponding integration layer of the CPU power model is depicted in Fig. 34.8

- The `apply_power_model` UPF command maps the run-time parameters in the CPU power model to corresponding signals of the CPU TLM.
- The same command is used to initialize the buildtime parameters in the CPU power model with static configuration parameters of the CPU TLM.
- The `add_edge_expression` command defines the conditions that cause a state transition in the power model.

In this simplistic example, the power model is triggered from a signal port of the TLM. This illustrates that the concept of UPF-3.0 IP power model is not limited to a system model but can be also applied to a RTL or gate-level representation of a component running in an HDL simulation, emulation, or FPGA prototyping environment. On the other hand, in a virtual prototyping environment, the transition of a power state can be triggered from all kinds of *observable events* in the TLM, e.g., the start or end of a transaction, a register access, an event on an analysis

```
::apply_power_model CPU -elements CPU0 -parameters \
  {  {vdd                          CORE0.VDD{V} } \
     {frequency                    CORE0.Frequency{MHz}} \
     {temperature                  CORE0.Temperature{C}} \
     {percent_active_power_in_wfi  /Power/percent_active_power_in_wfi} \
     {mA_per_MHz_at_1V             /Power/mA_per_MHz_at_1V} \
     {I_lkg_in_mA_at_1V_at_25C     /Power/I_lkg_in_mA_at_1V_at_25C{mA}} \
  }

::snps_upf::add_edge_expression -elements CPU0 -domain CORE \
    -state {OFF     -edge_expr {CORE0.VDD == 0} } \
    -state {WFI     -edge_expr {CORE0.wfi == 1} } \
    -state {ACTIVE  -edge_expr {CORE0.wfi == 0} }
```

Fig. 34.8 Example of context-dependent UPF-3.0 integration layer

instrumentation point, a specific software symbol executed on an Instruction-Set Simulator (ISS), etc. Due to the tool-specific nature of the edge expression, this command is so far not part of the official UPF standard and therefore implemented with a Synopsys-specific command.

Taken together, the context-independent power model and the context-dependent integration layer create a power analysis overlay model of a VP as depicted in Fig. 34.6.

34.2.3.3 Accuracy Considerations

The goal of system-level power analysis is not to provide 100% accurate power measurements but to replace high-level power estimation currently done with static spreadsheets. The actual accuracy of system-level power analysis depends mainly on the granularity of the power model and on the characterization of the power functions:

- The granularity of a SLP model is determined by the level of detail in the power model. For example, a CPU power model can be modeled as
  - a simple monolithic state machine as shown in the example above.
  - multiple domains for cores, cache, coprocessor, etc., each with their own state machine.
  - a detailed instruction-level power model, which calculates power based on the specific energy of each executed instruction.
- The characterization determines how the power expressions calculate the power consumption based on power estimates and/or measurements.
  - An early power characterization can be defined using high-level estimates, e.g., based on the extrapolation of power measurements from previous projects.
  - Once RTL and technology libraries are available, RTL or gate-level power estimation tools can be used to generate look-up tables, which determine power consumption based on design parameter configuration and operating mode.
  - Post-silicon power measurements can still be valuable to characterize the power consumption of a reusable IP block for usage in subsequent projects.

Despite the high level of abstraction, it turns out that VPs with system-level power analysis models provide power estimates in the order of 85–90% accuracy, which are good enough to steer architecture design decision and to guide software development in the right direction [8, 22].

34.2.4 Summary

This section provided an overview of the modeling methodologies enabling virtual prototyping. The first part surveyed the IEEE 1666 SystemC Transaction-Level Model (TLM) standard, emphasizing the Loosely Timed (LT) and Approximately

Timed (AT) modeling styles and their respective extensions. The second part gave an introduction to the new IEEE 1801-2015 UPF-3.0 modeling standard for system-level power analysis. The subsequent section elaborates on how these modeling techniques are applied to the creation and usage of VPs for early architecture analysis.

34.3 Virtual Prototyping for Architecture Design

Incorporating more and more functions and features into electronic products directly translates into increasing SoC design complexity. Devices integrate a multitude of heterogeneous programmable cores to achieve the necessary flexibility and power efficiency. The diverse communication requirements of all these cores lead to a complex interconnect and memory infrastructure to provide the required storage and communication bandwidth. For this purpose, the SoC interconnect and memory subsystem feature complex mechanisms like distributed memory, cascaded arbitration, and Quality of Service (QoS). As a result, dimensioning the SoC architecture, and in particular the interconnect and memory infrastructure, poses a variety of formidable design challenges:

Large Design Space
: Due to the complexity and configurability of the SoC infrastructure IP (interconnect, memory), tailoring the SoC infrastructure to the specific needs of the product requirements is a nontrivial task.

Dynamic Workload
: Multiple applications running at different points in time are sharing a limited set of available resources. Hence, the workload on the SoC architecture is difficult to estimate due to the multitude of product use cases.

High Price of Failure
: A weakly dimensioned SoC architecture leads to insufficient product performance (under-design) or excessive cost and power consumption (over-design). Both cases hamper the market opportunity of the final product.

High Potential for Optimization
: All the design decisions, which impact power, performance, and cost in a big way, need to be taken at the beginning of the development process.

This section first provides an introduction to virtual prototyping for architecture design and then dives into more detail about specific methods for architecture exploration, optimization, and validation.

34.3.1 Introduction

The introduction first discusses traditional methods for architecture design and then introduces a state of the art flow and modeling methodology based on a commercial virtual prototyping solution.

34.3.1.1 Traditional Methods

Architecture definition has always been a necessary step in any SoC design project. Traditionally, the performance has been analyzed using spreadsheets or detailed hardware simulation. However, the design complexity has reached a level where these methods are not appropriate anymore. On the one hand, static spreadsheet analysis does not take the dynamic behavior of multiple software applications and multiple levels of scheduling and arbitration in the executing hardware platform into account. This bears a great risk of mis-predicting the actual performance, which can lead to under- or over- design of the system architecture. On the other hand, hardware simulations are available late in the cycle, run very slow, and do not provide system-level performance analysis results. Hence, this is also not a suitable approach for early architecture analysis and optimization.

34.3.1.2 Virtual Prototyping Flow for Early Architecture Analysis

Synopsys Platform Architect for Multi-Core Optimization (MCO) is a virtual prototyping environment for early and accurate system-level performance analysis. This comprises libraries of simulation models for all relevant SoC components as well as tools for the assembly, simulation, and analysis of complete SoC platforms. A typical setup for memory subsystem performance analysis and optimization is shown in Fig. 34.9 below. The following paragraphs briefly describe the key elements in the model library and the architecture analysis flow. Please refer to [15] for a more complete introduction.

The flow to systematically analyze and optimize the architecture in Platform Architect is depicted in Fig. 34.9:

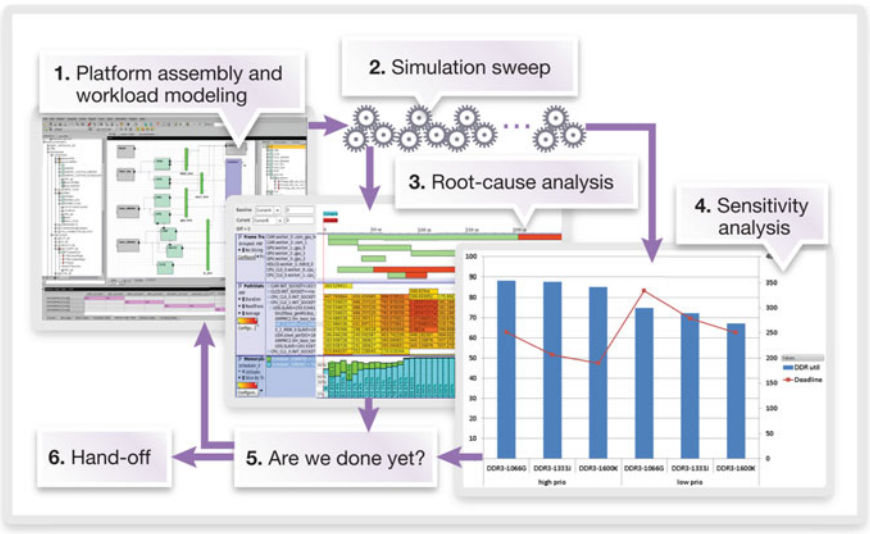

Fig. 34.9 Architecture analysis and optimization flow

1. Platform Assembly and Workload Modeling
   In the first step, the SoC performance model is assembled, connected, and configured in the Platform Architect authoring environment. This is quickly done based on the available libraries for workload, interconnect, and memory subsystem models.
2. Simulation Sweep
   Platform Architect generates a SystemC-based simulation of the SoC performance model. The simulation records a large variety of power and performance metrics into an analysis data base.
3. Performance Analysis
   The recorded data can be visualized and post-processed in the Platform Architect analysis tool. Various charts for throughput, latency, utilization, and contention allow the identification of performance issues. We can zoom on the time axis and further slice the results into the contribution from individual components for detailed root cause analysis.
4. Sensitivity Analysis
   Apart from single simulation runs, Platform Architect can also generate parameter sweeps, where a set of simulations is executed with user-defined configuration scenarios. This allows to systematically analyze the impact of the selected design parameters on high-level performance metrics. Each simulation result can be analyzed individually, but the results from all simulations are also aggregated into pivot chart tables for further post-processing in spreadsheet tools, e.g., Excel.
5. Are we done yet?
   Based on the analysis results, the architect needs to decide if the performance requirements in terms of throughput and latency cost are met. If this is confirmed, there might still be potential to improve utilization or to reduce contention in order to further optimize headroom, cost, or power consumption. In those cases, the iterative optimization loop continues by further modifying configuration parameters and setting up simulation sweeps until the design goals are reached.
6. Hand-off
   At the end of the optimization process, the final design configuration is handed off to the implementation team. As soon as the implementation becomes available, it can replace the system-level performance model with the RTL model. This allows the validation of the analysis results with the highest possible accuracy.

This generic iterative exploration and optimization flow can be applied to all kinds of architecture design problems. The next paragraph elaborates on how the architecture model should be constructed, depending on the specific objective of the architecture design project.

34.3.1.3 Modeling Methodologies for Early Architecture Analysis

The dimensioning of the SoC architecture is the first step in a design project. As depicted in Fig. 34.10, the input comes from the marketing requirements in terms of required features, supported features, performance numbers, and product cost.

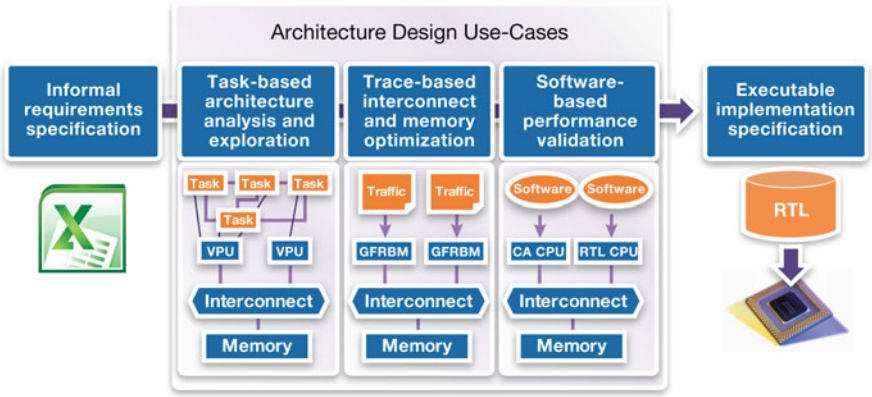

Fig. 34.10 Modeling methodology for different architecture design use cases

The outcome of the architecture design process is a detailed specification of the implementation. In between, three differentiated use cases have emerged over the last two decades of commercial deployment of architecture design methodologies.

All three steps include the modeling of the interconnect and memory architecture, because in many cases, this is the primary objective of the architecture design project. This is because individual IP blocks and subsystems can be developed independently, but when they are integrated into the SoC platform, the interconnect and memory subsystems need to satisfy the accumulated requirements from all IPs. Hence, the analysis of the interconnect and memory performance is typically the "common denominator" of all architecture design use cases.

The fundamental difference between the architecture design use cases is in how to model the remaining IP blocks and subsystems in the SoC:

- For the purpose of architecture validation, the actual software is executed on a fully functional and timing accurate VP, which is similar in nature to a VP for early software development: Instruction-Set Simulators (ISSs) are used for all programmable components, and functional and bit-accurate peripheral models are used for the nonprogrammable components. In addition, all models need to be enhanced with timing, which greatly increases the modeling effort and impacts the simulation speed.
- For the purpose of early architecture analysis and interconnect/memory optimization, the IP blocks are represented as abstract *workload models*. These task-based or trace-based workload models are nonfunctional and only represent the processing and communication requirements of each IP component, irrespective of whether they are programmable or not. The nonfunctional modeling of workloads is the key concept to achieve the necessary flexibility, modeling productivity, and simulation speed with sufficient temporal accuracy for early architecture analysis and exploration.

The following sections discuss the different architecture design use cases in more detail.

34.3.2 Software-Based Performance Validation

Historically, performance validation was the first commercially deployed architecture design use case. The idea is to build a functional and cycle-accurate VP of the complete SoC. The programmable IP subsystems require cycle-accurate representation of the CPU, capable of running the actual software. Depending on availability, this can be a Cycle Accurate (CA) SystemC TLM Instruction-Set Simulator (ISS) or the RTL of the CPU running in cosimulation or coemulation mode with the SystemC TLM platform. The nonprogrammable IP blocks need to be modeled in terms of CA SystemC TLMs. The benefit of such a fully accurate VP is that it allows the early validation of the final SoC performance. The analysis visibility into hardware and software enables the identification of performance issues and tuning of design and configuration parameters to optimize performance.

However, the overall return of investment (RoI) has proven to be challenging, especially for the growing complexity of complex many-core SoC platforms:

- Creating such a fully functional and cycle-accurate platform model requires a lot of initial modeling effort. Even if all models are available, it can be cumbersome to configure the software such that all the relevant traffic scenarios are covered.
- The simulation speed and the turnaround time for any change to the hardware or software is very slow.

For these reasons, performance validation is typically done using emulation or hardware prototyping methods. A new trend is hybrid emulation and prototyping, which allows to combine the best of both worlds:

- Leverage emulation and hardware prototyping for large IP blocks, e.g., CPU, GPU, and custom IP, to achieve reasonable simulation speed and avoid the effort to create cycle-accurate models.
- Leverage virtual prototyping for interconnect and memory subsystem to analyze and optimize performance critical parameters with high analysis visibility and fast turnaround time.

Architecture analysis with VPs starts very early in the development process. At this point, neither the software nor the RTL of the major IP blocks is available. Therefore, initial performance analysis is carried out using workload models instead of real software. As the platform specification and implementation matures, the workload models can be incrementally replaced by the cycle-accurate functional models or the RTL. This gradually converts the architecture exploration model on the left side of Fig. 34.10 into a cycle-accurate VP as depicted on the right. This way, the initial assumptions in the workload model can be validated, and the architecture can be fine tuned.

The following sections describe in more detail the creation of VPs for architecture analysis using trace-based and task-based workload modeling.

34.3.3 Trace-Based Interconnect and Memory Optimization

In many cases, the interconnect and memory optimization is the key concern of the architecture definition phase. For this purpose, it is often sufficient to model the workload of all relevant initiator components in terms of transaction trace files, which are replayed by Generic File Reader Bus Masters (GFRBMs). Trace-based workload modeling has proven to be the most productive methodology to quickly and accurately analyze and optimize this critical part of the SoC architecture. Traces are available early in the development process, long before the actual software exists. They can be easily recorded or generated, they execute fast, and they represent the transaction sequence and timing of the real application with sufficient accuracy.

34.3.3.1 Traffic Generation

A GFRBM is sufficient to model the traffic generated by all kinds of initiator components like CPU, GPU, DMA, etc. The critical portion is the creation of elastic traces, which accurately represent the actual traffic of the corresponding initiator component. Here elasticity refers to the requirement, that the trace generator needs to respond to a change in the interconnect and memory architecture in the same way as the corresponding initiator component. An important aspect of elasticity is the synchronization of different traffic flows: If one traffic flow is triggered by another flow, then this dependency needs to be explicitly represented in the trace-based workload model. An example of such an elastic trace is shown in Fig. 34.11.

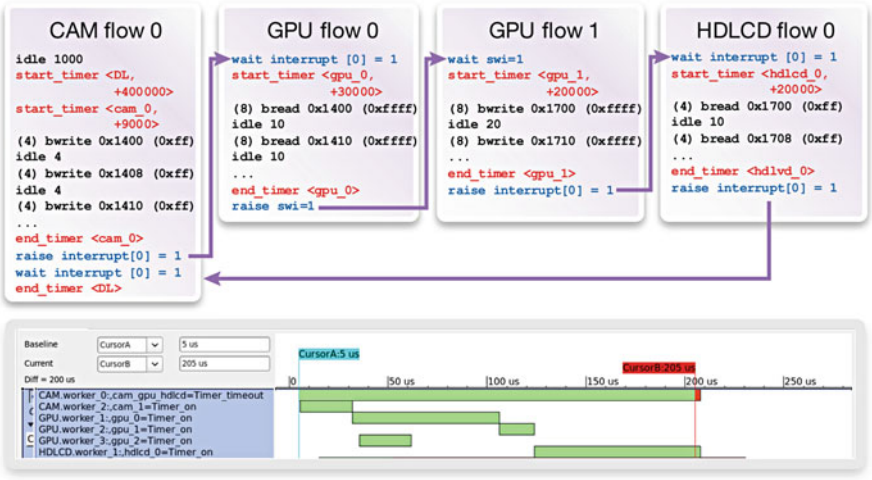

Fig. 34.11 Elastic trace-based workload model (*top*) with deadline analysis (*bottom*)

The upper part of Fig. 34.11 shows an example of the transaction-level trace format that is executed by the GFRBM. Each traffic flow is described as a sequence of reads and writes with the relevant transaction attributes, e.g., burst size, command, address, byte enables, etc. As opposed to using absolute time stamps, the idle command defines the relative number of cycles between two subsequent transactions.

Multiple traffic flows can be synchronized by using the raise and wait for internal and external interrupt signals. In the example above, flow 0 on the GPU only starts after flow 0 on the camera raises interrupt 0. In the same way, flow 1 on the GPU waits for flow 0 to finish.

It is also important to capture performance constraints of traffic flows. For example, the `cam_0` timer in Fig. 34.11 expires after 9000 cycles to indicate that the transaction sequence should be processed before this deadline. The lower part of Fig. 34.11 shows the visualization of the deadlines: The end-to-end constraint of the camera, GPU, and HD-LCD starts at 5 µs and turns red at 205 µs, indicating that the 200 µs deadline is just missed.

34.3.3.2 Transaction-Level Models for Interconnect and Memory Subsystem

Obviously the key ingredient for analyzing and optimizing the performance of the interconnect and memory subsystem are sufficiently accurate models of these components. Commercially available virtual prototyping environments provide libraries of SystemC TLMs at different levels of abstraction.

- Highly configurable approximately timed models, which can be used to mimic a specific IP.
- Highly accurate model of a specific IP, which represent all relevant design and configuration parameters

An example of a configurable approximately timed model is the generic Multi-Port Memory Controller provided in the Synopsys Platform Architect model library. This memory controller model is based on the Fast Timed (FT) TLM protocol to support multiple bus protocols (see paragraph on Extended AT in Sect. 34.2.1). It incorporates the features of modern memory controllers, e.g., transaction re-ordering, address mapping, configurable number of ports, buffer sizes, frequency ratio, and QoS. This also includes a timing model of all commonly used Double Data Rate (DDR) standards (DDR2, DDR3, DDR4, mDDR, LPDDR2, LPDDR3, LPDDR4, DDR3-3DS) with their respective speed bins and device types [16]. An example of an accurate IP-specific model for architecture analysis is the Architects View (AV) TLM model of the FlexNoC interconnect from Arteris [19]. The port interfaces of the AV FlexNoC model use an extended TLM-2.0 AT protocol, which accurately represents the FlexNoC NTTP protocol with extended attributes and phases, but which is also compliant with the AT-BP (see Sect. 34.2.1). Internally the AV FlexNoC model represents all the performance-relevant aspects of the NoC architecture, like the topology, serialization, clock schemes, buffering, arbitration

schemes, pipeline stages, and transaction contexts. This also includes advanced QoS schemes like run-time bandwidth regulation. The FlexNoC model is generated from the NoC configuration tool, which is later used to generate the actual RTL implementation. This way, the optimized interconnect configuration is seamlessly used for implementation. The model library from Synopsys Platform Architect contains similar models for other popular interconnect IP, e.g., ARM CoreLink NIC-400.

Based on these kinds of available TLMs for architecture analysis, it is very little effort to assemble a performance model of any SoC platform, which allows to analyze and optimize the memory subsystem [17, 20, 24].

34.3.4 Task-Based Architecture Analysis and Exploration

The most recent commercially deployed architecture design methodology is the early exploration and optimization of complex Multi-Processor System-on-Chip (MPSoC) platforms using task-based workload models. This allows the quantitative analysis of performance and power metrics to avoid SoC market failure due to underperforming or overly power hungry architectures. The key architecture questions that SoC hardware architects can analyze are:

- How to partition the application into fixed HW accelerators and software executing on processors?
- What is the optimal number and type of CPUs, GPUs, Digital Signal Processors (DSPs), and HW accelerators?
- How to dimension the interconnect and memory architecture?
- What is the expected performance/power curve?

34.3.4.1 Modeling Methodology

As depicted in Fig. 34.12, the modeling methodology for task-based architecture analysis follows the Polis Y-chart approach [5], similar to the framework described in the ▸ Chap. 9, “Scenario-Based Design Space Exploration”.

- SoC application workloads such as CPU load, imaging, video encoding and decoding, modem, and network packet processing are represented as an application task graph.
- The VP of the SoC platform contains all relevant processing elements as well as interconnect and memory resources. The key component is the Virtual Processing Unit (VPU), which represents all kinds processing elements such as CPUs, GPUs, DSPs, and HW accelerators. VPUs are high-level processor models that execute the portion of the task graph [13, 18].
- The task-based application workload model is mapped to the architecture model to construct an executable specification of the application running on the hardware platform.

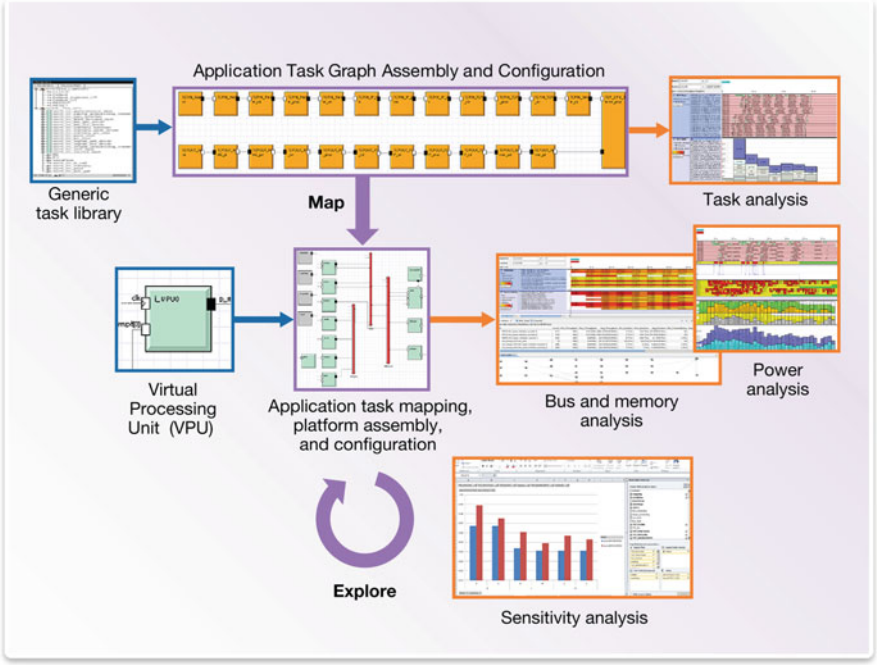

Fig. 34.12 Early architecture analysis with task-based workload models

The following sections describe the different aspects of application modeling, platform modeling, and mapping in more detail.

34.3.4.2 Task-Based Workload Models

In general, a task-based workload model captures the processing and communication requirements of the application. As showcased in Fig. 34.13, the overall application is broken down into a set of tasks, which exhibits the available coarse-grained parallelism. The connections in the task graph denote the execution precedence, e.g., in this example Task C and D execute after B. In addition, each task is characterized with a set of processing- and communication-related parameters. A typical set of parameters is depicted in Fig. 34.13:

- A source task like Task A, the `wait_cycles` parameter specifies the delay between two consecutive activations
- The `processing_cycles` specifies minimum number of cycles for which a task occupies a resource.
- The `load_ratio` `store_ratio` specify the additional communication overhead, e.g., on average Task B generates 50 load and store transactions per activation.

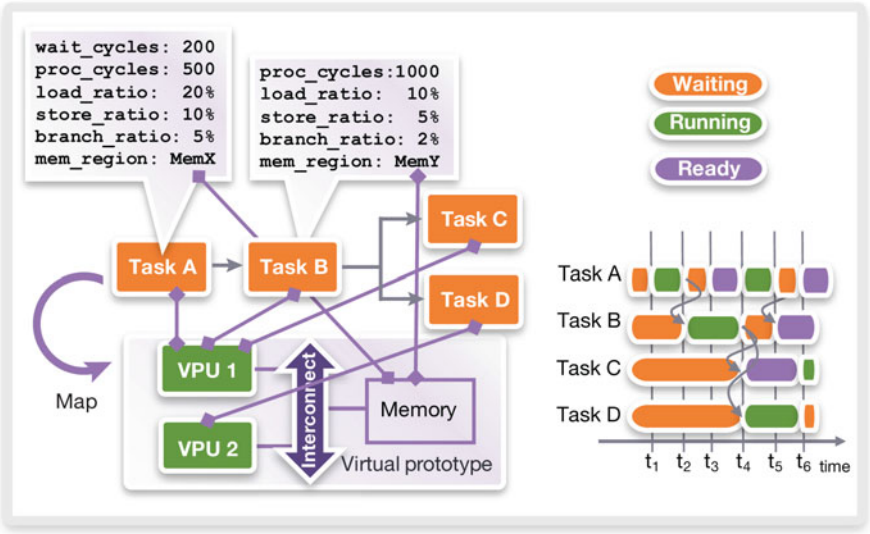

Fig. 34.13 Specification, mapping, and execution of an application task graph

- The `branch_ratio` can be used to model jumps in the address sequence and additional processing overhead for mis-predicted branches.
- The `mem_region` defines a logical name for the memory location.

In the mapping step, tasks are assigned to VPUs, which represent the execution resources, and the logical memory regions assigned to physical memories in the platform.

From the modeling perspective, tasks and connections are realized on top of standard SystemC concepts like threads and events. In addition, tasks have explicit states like *Created*, *Ready*, *Running*, *Waiting*, *Suspended*, and the ability to *consume* processing time. This enables the modeling of software processes executing in the context of an Operating System (OS). The task state trace on the right side of Fig. 34.13 showcases the execution of the given example task graph:

- In the beginning, all task are in state *Waiting*
- At t_1, the wait cycles of source Task A are expired. VPU 1 is available, so Task A changes immediately into state *Running*.
- At t_2, the processing cycles of Task A are finished. Task A triggers Task B, which changes into state *Running*.
- At t_3, the wait cycles of Task A are again expired, but this time VPU 1 is still occupied with Task B, so Task A transitions into state *Ready*.
- At t_4, Task B is done, so Task A becomes *Running* on VPU 1. Also, Task B triggers Task C and D. C is blocked by Task A on VPU 1 and remains *Ready*, but Task D can immediately transition into state *Running* on VPU 2.

In reality, more complex scheduling algorithms with priorities, preemption, time slices, etc., influence the execution pattern. The actual duration a task remains in state *Running* is further impacted by dynamic effects like bus arbitration and dynamic memory latencies.

Synopsys provides a generic task library with a set of configuration parameters, so users can rapidly compose a task graph without manual modeling effort. This library provides a set of generic configurable tasks to create a nonfunctional performance model of arbitrary application topology.

For data processing applications, such as audio, video, networking, or wireless communications, it is typically straightforward to define the application task graph using the elements in the generic task library. For control-oriented applications, e.g., in the automotive domain, and for higher-level applications running on top of an OS, the application task graph can be automatically generated from software execution traces.

34.3.4.3 Performance Model of the System-on-Chip

Multi-core architectures are composed of SystemC TLMs, such as interconnect, memory controller, DMAs, and other components which are available in the Platform Architect model library. The VPU models the processing elements (CPUs, GPUs, DSPs, and HW accelerators), which can execute a task graph, or a portion of the task graph. The VPU task scheduler supports preemption and time slicing of tasks for modeling of interrupts and arbitrary OS scheduling algorithms. The provided set of default scheduling algorithms can be extended by the user. The VPU also comes with a library of components for traffic generation, cache modeling, inter-VPU communication, and interrupt handling to model the realistic execution of a task graph with sufficient accuracy.

34.3.4.4 Application to Architecture Mapping

The next step is to map the application task graph onto the VPUs. This way the tasks are assigned to a physical execution resource, and the logical memory regions in the task graph are mapped to physical memories in the platform. The number of processing resources per VPU is configurable and determines how many tasks can run in parallel. For example, a VPU with four resources represents a Symmetric Multi-Processing (SMP) cluster with four cores. VPUs with different parameters (scheduling algorithm, clock period, traffic generation, etc.) represent an asynchronous multiprocessor (AMP) subsystem. A VPU with only one task mapped to it can represent a dedicated hardware block.

The result is an executable system performance model, which simulates the execution of a task-based workload model on a resource constraint platform. The analysis monitors measure a variety of performance metrics like latency, throughput, utilization, and contention. This way, an SoC architect can identify performance issues, bottlenecks, or underutilization of resources [8].

34.3.4.5 Joint Power and Performance Analysis

The availability of the IEEE 1801-2015 UPF-3.0 standard for system-level power analysis [11] enables the early estimation of the system power consumption by adding power monitors as an overlay to a VP for performance analysis; see Sect. 34.2.3 and Fig. 34.6). Compared to static power analysis based on spreadsheet, this provides much more realistic power estimates, because the dynamic activity of each component in the platform is taken into account. This way, architects can analyze the impact of architecture design decision on the power consumption [9]. Many state-of-the-art SoC platforms optimize the power consumption using run-time power management schemes, e.g., clock gating, power gating, and Dynamic Voltage and Frequency Scaling (DVFS) [10]. These power management schemes bear great potential to reduce power consumption, but they also come at additional cost. Hence power management adds another dimension to the architecture design space. Unfortunately, power consumption and performance cannot be considered in isolation. For example, reducing voltage and frequency reduces power but increases execution time. The resulting impact on the energy consumption is not obvious, so without quantitative analysis, it is difficult to decide between power management strategies like Run Fast Then Stop (RFTS) or DVFS.

As depicted in Fig. 34.14, adding a model of the power management to a VP for architecture analysis enables the quantitative analysis of power management strategies, including the impact of power management on power and performance:

- How should the SoC be partitioned into DVFS domains to best serve the target application use cases?
- Based on the activity profile of a component, do the power savings justify the additional cost for applying clock gating or power gating or both?

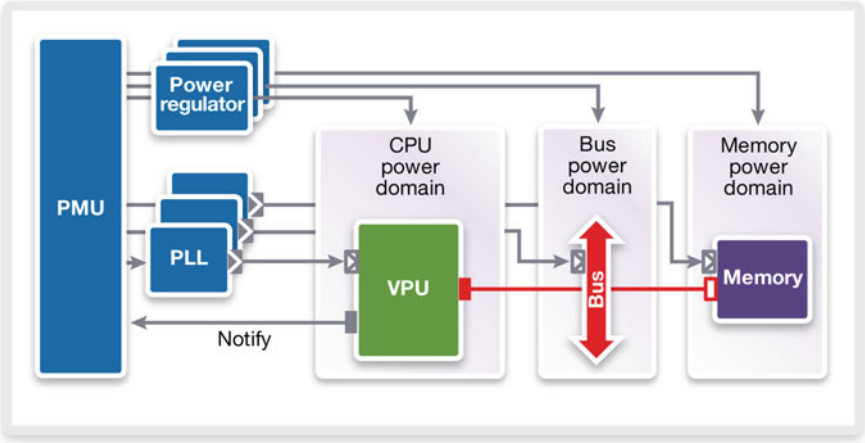

Fig. 34.14 DVFS modeling for joint power and performance analysis

- How many DVFS operating points are needed to effectively reduce power consumption?
- How aggressive can the frequency be reduced before application violates real-time requirements?

Figure 34.14 shows how to model the impact of DVFS and power management in the context of a multi-core platform in Platform Architect for MCO. A functional model of the Power Management Unit (PMU) is part of the SoC platform model to take the performance and power aspect into account. The processing elements (the VPUs) notify the PMU when they become active or idle. In response to this, the PMU model controls the frequency of the processing elements and the voltage levels of the power supply regulators. This captures the impact of the DVFS power management on the performance: The execution time of the tasks running on the VPU depends on the actual frequency. The same task takes longer when the clock is running at a lower frequency. The frequency and voltage levels are also used as run-time parameters in the UPF-3.0 power models to measure actual power consumption.

The outcome of the early analysis of power and performance is an optimized power architecture:

- An optimized specification of the system-level power intent, including the power supply architecture and the grouping of the SoC into power domains.
- The definition of the most promising power management policies.
- A realistic estimation of the system-level power and energy consumption for a given workload.

The UPF standard can also be used to export the resulting optimized system-level power intent from the virtual prototyping environment to the subsequent implementation and verification tools.

34.4 Conclusions

This section provided an overview of virtual prototyping for architecture design. It introduced software-based performance validation, trace-based interconnect/memory optimization, and task-based architecture exploration as the three main architecture design use cases, which are in commercial deployment today. Especially the joint power and performance analysis based on task-based application workload models sees growing adoption, because it is the most effective approach to cope with the "complexity wall" [26] under the given competitive landscape and time-to-market pressures.

Acknowledgments The author acknowledges Tom De Schutter for contributing the sections on software development and on system validation to the introduction of this chapter as well as Alan Gibbons for contributing the section on system level power analysis to the introduction of this chapter.

References

1. AMBA AXI and ACE Protocol Specification (2013). http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html
2. AMBA-PV Extensions to TLM Developer Guide (2015). http://infocenter.arm.com/help/topic/com.arm.doc.dui0846f/DUI0846F_ambapv_extensions_to_tlm_2-0_dg.pdf
3. ARM Fast Models. http://www.arm.com/products/tools/models/fast-models
4. Bailey B et al (eds) (2010) TLM-driven design and verification methodology. Cadence Design Systems, San Jose. https://www.synopsys.com/vpbook
5. Balarin F et al (1997) Hardware-software co-design of embedded systems: the POLIS approach. Kluwer Academic Publishers, Boston
6. Barroso L, Holzle U (2007) The case for energy-proportional computing. Computer 40(12): 33–37. doi: 10.1109/MC.2007.443
7. Datta S, Bonnet C, Nikaein N (2012) Android power management: current and future trends. In: 2012 first IEEE workshop on enabling technologies for smartphone and internet of things (ETSIoT), pp 48–53. doi: 10.1109/ETSIoT.2012.6311253
8. Fadi Aboud IC (2014) Balancing power performance and user experience using virtual prototyping. In: Proceedings of the synopsys users group conference (SNUG), Israel. https://www.synopsys.com/news/pubs/snug/2014/israel/B2_Aboud_pres_user.pdf
9. Fischer B, Cech C, Muhr H (2014) Power modeling and analysis in early design phases. In: Proceedings of design, automation and test in Europe conference and exhibition (DATE), pp 1–6. doi: 10.7873/DATE.2014.210
10. Flynn D, Aitken R, Gibbons A, Shi K (2007) Low power methodology manual for system-on-chip design. Springer, New York
11. IEEE Standard for Design and Verification of Low-Power Integrated Circuits (2015). http://standards.ieee.org/getieee/1801/download/1801-2015.pdf
12. IEEE Standard for Standard SystemC Language Reference Manual (2011). http://standards.ieee.org/getieee/1666/download/1666-2011.pdf
13. Kempf T, Doerper M, Leupers R, Ascheid G, Meyr H, Kogel T, Vanthournout B (2005) A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms. In: Proceedings of design, automation and test in Europe, vol 2, pp 876–881. doi: 10.1109/DATE.2005.21
14. Kogel T (2006) Peripheral modeling for platform driven ESL design. In: Burton M, Morawiec A (eds) Platform based design at the electronic system level. Springer, Dordrecht, pp 71–85
15. Kogel T (2013) Designing the right architecture, SoC interconnect and memory optimization with synopsys platform architect. Synopsys whitepaper. https://www.synopsys.com/cgi-bin/proto/pdfdla/pdfr1.cgi?file=pa_soc_v4_wp.pdf
16. Kogel T (2016) Optimizing DDR memory subsystem efficiency, Part 1: the unpredictable memory bottleneck. Synopsys whitepaper. https://www.synopsys.com/cgi-bin/proto/pdfdla/pdfr1.cgi?file=optimizing-ddr-efficiency-p1-wp.pdf
17. Kogel T (2016) Optimizing DDR memory subsystem efficiency, Part 2: case study. Synopsys whitepaper. https://www.synopsys.com/cgi-bin/proto/pdfdla/pdfr1.cgi?file=optimizing-ddr-efficiency-p2-wp.pdf
18. Kogel T et al (2005) Integrated system-level modeling of network-on-chip enabled multi-processor platforms. Springer, Dordrecht
19. Lecler J-J, Baillieu G (2011) Application driven network-on-chip architecture exploration & refinement for a complex SoC. Des Autom Embed Syst 15(2):133–158. doi: 10.1007/s10617-011-9075-5
20. Patel S, Sood B, Semiconductor F (2014) Quick, re-usable and cost effective approach to create accurate models using synopsys platform architect framework for early system level performance analysis. In: Proceedings of the synopsys users group conference (SNUG), India. http://www.synopsys.com/news/pubs/snug/2014/India/paper_sood.pdf

21. Reyes V (2012) Virtualized fault injection methods in the context of the ISO 26262 standard. SAE Int J Passenger Cars Electron Electr Syst 5(1):9–16
22. Schurmans S, Zhang D, Auras D, Leupers R, Ascheid G, Chen X, Wang L (2013) Creation of ESL power models for communication architectures using automatic calibration. In: 2013 50th ACM/EDAC/IEEE design automation conference (DAC), pp 1–6. doi: 10.1145/2463209.2488804
23. Schutter TD (ed) (2014) Better software. Faster! best practices in virtual prototyping. Synopsys Press. https://www.synopsys.com/vpbook
24. Skrzeszewski TK, Intel Corp. (2015) ATOM mobile SoC performance and power architecture exploration. In: Synopsys users group conference (SNUG), Santa Clara. http://www.synopsys.com/news/pubs/snug/2015/silicon-valley/mb08_skrzeszewski_paper.pdf
25. Synopsys DW TLM library. http://www.synopsys.com/Prototyping/VirtualPrototyping/VPModels/Pages/DW-TLM-Library.aspx
26. Teich J (2012) Hardware/software codesign: the past, the present, and predicting the future. Proc IEEE 100(Special Centennial Issue):1411–1430. doi: 10.1109/JPROC.2011.2182009
27. The SystemC Modeling Library (SCML). http://www.synopsys.com/cgi-bin/slcw/kits/reg.cgi
28. Tool Command Language (TCL). http://www.tcl.tk