COMPLEX - COdesign and power Management in PLaform-based design space EXPloration

(Invited Paper)

Kim Grütner, Philipp A. Hartmann, Kai Hylla, Sven Rosinger*, Wolfgang Nebel†
Fernando Herrera, Eugenio Villard, Carlo Brandolesi§, William Formaciari$, Gianluca Palermo$
Chantal Ykman-Couvreur*, Davide Quaglia‖, Francisco Ferrero, Raúl Valencia**

*OFFIS – Institute for Information Technology, Oldenburg, Germany, Email: {gruetner | hartmann}@offis.de
† Carl von Ossietzky University Oldenburg, Germany, Email: nebel@informa.uk.uni-oldenburg.de
‡ Universidad de Cantabria, Santander, Cantabria, Spain, Email: {herrera | evillar}@teisa.unican.es
§ Politecnico di Milano, Italy, Email: {brandole | formaci | gpalermo}@elet.polimi.it
$ IMEC, Belgium, Email: ykman@imec.be
‖ EDA Lab s.r.l. - Networked Embedded Systems, Verona, Italy, Email: davide.quaglia@edalab.it
** GMV Aerospace and Defence S.A.U - Madrid, Spain, Email: {ferrero | rvalencia}@gmv.com

Abstract—The consideration of an embedded device's power consumption and its management is increasingly important nowadays. Currently, it is not easily possible to integrate power information already during the platform exploration phase. In this paper, we discuss the design challenges of today's heterogeneous HW/SW systems regarding power and complexity, both for platform vendors as well as system integrators.

As a result, we propose a design flow concept that combines system-level power optimization techniques with platform-based rapid prototyping. Virtual executable prototypes are generated from MARTE/UML and functional C/C++ descriptions, which then allows to study different platforms, mapping alternatives and power management strategies.

Our proposed flow combines system-level timing and power estimation techniques available in commercial tools with platform-based rapid prototyping. We propose an efficient code annotation technique for timing and power properties that enables fast host execution as well as adaptive collection of power traces. Combined with a flexible design-space exploration (DSE) approach our flow allows a trade-off between different platforms, mapping alternatives, and optimization techniques, based on domain-specific workload scenarios. The proposed flow is currently under implementation in the COMPLEX FP7 European integrated project.

I. INTRODUCTION

Increasing computing power and shrinking sizes of processing elements allow more and more functionality to be realized within embedded systems. In earlier times embedded systems have only implemented simple functions, but today complete systems, also known as System-on-a-Chip (SoC), can be implemented on a single chip. In the past a lot of effort has been spent on characterizing and estimating hard- as well as software parts of the SoC. For both of them relatively good techniques and tools exist, hence their properties and characteristics are well known and well understood.

In large and complex SoCs, components of the system, such as hardware, software, memory, and interconnect, cannot be considered separately. They must be considered together while interacting to capture the entire system behavior. For analysis of the dynamic behavior this implies a behavioral simulation of all system components and their interaction. This is essential for power and timing estimations. For specific platforms proprietary simulation environments are available for both timing and power models. But a common and open framework, suitable for a large range of platforms and designs is still missing. Such a framework would allow comparing different platform characteristics and thus rapid prototyping and design space exploration. Performance bottlenecks and power peaks within the entire system could be identified in early design phases, where modifications of the system are easier and more affordable than in later phases.

In this paper we present a new design flow concept and propose a framework that combines system-level power optimization techniques with platform-based rapid prototyping. For the proposed approach we focus on the derivation of an executable virtual prototype of an embedded HW/SW systems right after functional specification, partitioning, and mapping to an execution platform. Execution or simulations of this prototype allow decisions concerning architecture, performance, power, and price, early in the development process. The challenges we need to face in today's Electronic System Level (ESL) design-flow, including existing point tools that are able to address single design issues, are presented in Section II. Our proposed framework follows a unified system-level specification for HW and SW, but utilizes different estimation techniques for custom HW and SW, as well as pre-defined IP components. Since good stand-alone solutions for each of these parts exist, Section III presents our concept that unifies state-of-the-art tools and techniques in a common design-flow. This work closes with a conclusion in Section IV, which sums up our concept and points out following work.

II. REQUIREMENTS & EXISTING TOOLS

This section aims at defining requirements for early system specification, modeling, and simulation at ESL. Moreover, we are giving a brief overview of commercial tools that partly fulfill these identified requirements.
Generally, embedded system design is a sequence of design decisions that finally leads to an implementation. These decisions are made with regard to the functional and extra-functional system requirements. These need to be captured and tracked during the entire design flow. All dimensions (e.g., functional, timing, power, performance, memory, area, etc.) of the design space need to be explored in order to estimate the costs of the final system implementation. For this purpose a dependable and accurate model of the future system and workload scenario, prior to heavy investments in HW and SW development is necessary. In our approach we focus on the enrichment of virtual execution platforms with extra-functional properties. To limit the proposed approach we do not consider a direct path to implementation or automatic synthesis of physical platforms, yet.

Today’s virtual prototyping platforms are provided for early software development and functional testing. These platforms are often built using OSCI TLM2 interfaces at PV (Programmers View). Today these platforms are used for functional software development and software stack configuration. Most virtual platforms are targeting mainly functional and loose timing aspects in early software development. Power dissipation or the functional effects of dynamic power management strategies cannot be taken into consideration, and thus cannot be explored at early design stages.

Code generation in Model-Driven Engineering (MDE) high-level synthesis techniques for custom HW, and abstraction of existing components to system-level descriptions are key enablers for unified system-level simulation approaches [1] and need to be urgently combined with the virtual prototyping platform approach as mentioned above. We want to consider functional, power and timing behavior on system-level under explicit partitioning and mapping to a specific implementation platform. At first, this requires a formal system-level specification to be linked with functional and extra-functional requirements to be checked during the entire design process. Secondly, it requires an executable model that can be derived automatically from the formal specification model, which is capable to simulate large sequences of domain-specific workloads in an acceptable time. Thirdly, the representation and annotation of extra-functional information (timing and power) in the executable specification. For today’s embedded HW/SW systems this cannot be done on RT-level anymore due to high simulation complexity. The requirements of the proposed framework, along with point-tools (as described in [2]) that fulfill some of these requirements, are subdivided into four categories discussed below:

A. MDE and Executable Specification

The complexity and size of current systems are increasing which may well lead to longer development times and integration challenges. In the contrary, time-to-market constraints are tighter and there is a larger incidence of extra-functional demands (i.e. power consumption, real-time behavior). Therefore, it is necessary to provide mechanisms that permit the separation of the functional and extra-functional concerns in order to better master the complexity of current embedded systems. The adoption of the Model-Driven Engineering (MDE) principles improves not only how functional and extra-functional concerns are managed, but also allow the designer to parallelize the specification of both the HW and SW systems. The designer should operate on a parallel executable specification that enables the same functional description for HW and SW. This is permitted by definition of an common specification language at system level, a unique, well-defined parallel programming paradigm and the formulation of HW/SW partitioning through different platform mappings. The communication model should be abstracted from platform communication resources and enable the remapping to different communication infrastructures.

Following a component-based approach is highly desirable in a HW/SW design as it improves the product organization, its reusability and modularity. In this approach, the component represents an elemental unit for architectural mappings, which simplifies the analysis of the system extra-functional attributes and facilitates the design space exploration.

The system specification follows the platform-based design (PBD) approach [3], [4]. According to the MDE principles the specification of the system functionality and the platform should be done separately. The functional properties of the system should be specified in the application specification. Extra-functional properties like the deadline of application tasks or the use of different IP cores, target technology, maximum chip area, etc. which influence timing and power dissipation need to be specified separately. Some widely used commercial available tools and languages for parallel executable system specification are: LabView, Matlab/Simulink & Stateflow, Rhapsody & Statemate, Esterel SCADE, and C/C++/SystemC-based approaches. For adding extra-functional properties different UML profiles, like MARTE, have been proposed. All of these tools, modeling languages and profiles already offer a direct path to implementation for embedded HW/SW systems. However, the effects of these implementations in terms of timing behavior and power consumption can hardly be analyzed. [5] proposes an interesting approach to power efficient system design using UML generated executable models. The use of the MDE principles in the design of HW/SW systems and standard modeling languages like UML allows the definition of modeling methodologies that are based on the separation of functional and extra-functional concerns and the specification of both hardware and software system. This is convenient in a context where system engineers need to produce a specification of the embedded systems suitable for early design stages (e.g. functional verification, design space exploration). Indeed, it can also serve as an unambiguous and easily understandable contract for hardware and software designers in later phases of the design.

Moreover, the use of modeling languages like UML provides an unambiguous way of documenting the system architecture (i.e. both hardware and software) where a set of diagrams provide a mean to illustrate the different system facets that are for the interest of the designer. Through extensive
tool support, from UML model editors to model-to-model and model-to-text, the automatic generation of system executable specifications is enabled. As it was mentioned above, such automation not only saves time, but also leverages a unified approach once the coherence between the UML model and the executable model can be better ensured.

However, UML itself does not provide the necessary semantics to model extra-functional properties. The use of the MARTE profile gives some of the required semantics (i.e., target technology, area, task deadlines), while others related to the DSE must be provided by means of UML profiling mechanisms.

Finally, modern dynamic performance estimation techniques (i.e., native simulation, virtualization) enable the generation of fast executable performance models, suitable for feeding a design exploration loop.

B. Estimation and Representation of Extra-Functional Properties

Following the PBD approach enables a separation of application and execution platform. The mapping describes a binding of the application to allocated execution, communication, and memory resources of the platform. All these mapping decisions have an influence on the system properties including the correct behavior in terms of timing, power, and costs.

In this paper we focus on timing and power dissipation aspects, referred to as implementation artifacts of the execution platform, consisting of processing elements, memories, and interconnect. As mentioned in the previous paragraph, several tools exist that are able to generate a C/C++/SystemC implementation of the behavior of the system. This representation shall be a common starting point to perform analysis and code optimizations under consideration of constraints induced by the chosen target platform and user-defined parameters.

The consideration of the system's timing and power dissipation needs to be studied under domain specific workload scenarios. For the assessment of the system's overall power dissipation, average or typical cases are usually more important than worst case workloads. Concerning real-time requirements worst-case execution times (WCET) need to be studied instead of average case execution times. For an overall system characterization both kinds of analysis need to be performed for the software, custom hardware, communication, and pre-existing IP components.

Power and timing estimation of full-custom HW (ASIC) depends on several design parameters like micro-architecture size and design, level of parallel processing, layout implications but it also depends on runtime characteristics like activity, frequency, or dynamic power management policies. Transistor technology also has major impact on dynamic and static power values as well as on timing. Additionally, the environmental temperature has major impact on HW's power and in extreme case may force the chip to slow down. The timing and power estimation of ASIC hardware under consideration of these aspects have gained in importance and are addressed by today's industrial high-level synthesis tools, such as Mentor Graphics' Catapult C.

Estimation of embedded software's power and timing is complicated due to processor attributes like pipelining, branch prediction, superscalar architectures, and out of order execution. Additionally, power and timing of the processor depends on external influences such as memory stalls, interrupts, task switches, and operation system calls. To cover these aspects during power estimation vendor specific or generic instruction set simulators (ISS) as ARMulator or SimpleScalar with augmented power models are used. For SW WCET analysis static analytical methods are used instead of simulative approaches. AbsInt's aIT WCET analysis tool is used by the industry and allows easy integration with existing executable specification environments like Esterel's SCADE Suite.

Estimation of extra-functional properties for communication depends on the implemented functionality, the partitioning into embedded SW and custom HW and its mapping to execution units, and finally the data dependencies induced by the provided use-case. Using the SystemC-TLM methodology allows vendor specific provision of bus models. These models can be implemented on different levels of granularity, from abstract atomic transitions, over the consideration of request and transfer phases, down to a bus-cycle accurate view. Timing of communication should be encapsulated in dedicated communication models and should reflect the communication protocol's behavior as well as data dependent effects, such as transaction lengths. For an appropriate communication power model the situation is more difficult, since the vendor of the bus model does not necessarily have information about its implementation and its influencing parameters for the power model. To this end the bus vendor will not be able to provide a power model, but the API of the bus-functional model shall offer extension points for the platform developer.

A realistic system contains all these types of components. They must be considered together to observe the influences of their interaction. Thus, a holistic estimation combines individual estimation results for embedded SW, custom HW, and communication, to allow a simulation and estimation of the overall system behavior and its extra-functional properties. Such an approach uses existing and highly specialized tools to characterize individual parts of the system. Typical estimation approaches, as mentioned above, provide models for the underlying HW. The challenge is to use information from these low-level models for creating higher-level models that can be combined with the behavior in a functional simulation.

C. System Simulation Including Extra-Functional Properties

Fast system simulation is important due to the raising system complexity. Today most state-of-the-art mobile embedded systems like phones, cameras, music and video players are using complex HW and SW, including operating systems and complex middleware. Virtual prototypes that are used for the simulation of these systems can no longer be executed on a cycle-accurate ISS connected via a cycle accurate bus model to a register transfer (RT) level representation of custom HW.
To boost simulation performance virtual platform simulators allow to abstract from certain HW platform details, like the cycle-accurate representation of a processors instruction set or cycle accurate communication models. Furthermore, today's virtual platform simulators make use of native host system execution to speed-up simulation. For the representation of extra-functional properties some approaches are using source annotated execution time and power information from static code analysis. This approach allows a replacement of the instruction set simulation model and removes a major simulation speed bottleneck. Nevertheless, the raise of abstraction and its gain in execution speed comes at the cost of a loss in accuracy.

But depending on the application characteristics, workload scenarios, and of course the system issue under investigation, a certain loss in accuracy is acceptable. A simulation framework should offer the possibility to switch between different simulation granularities for both power and timing. Commercial available simulation platforms are ARM RealView & MaxSim, Mentor Graphics Vista, Synopsys Virtualizer & CoMET and METeoR, Cadence System Development Suite, and Wind River Simics.

D. Design-Space Exploration

Platform-based design approach is widely used to meet time-to-market constraints. In this scenario, the basis for designing novel systems is to have platform with lots of parameters that can be exposed to the designer to enable finding the best HW/SW architecture that meets the application requirements. The tuning phase of platform configuration, hardware-software partitioning, scheduling, and mapping on the platform resources is called Design Space Exploration (DSE).

In the past, this phase has been mainly done by using manual approaches guided by the designer experience. Nowadays this possibility is becoming unfeasible for two main reasons: the first deals with the time needed for the exploration phase, while the second one with the performance unpredictability of novel parallel architectures.

Regarding the first reason, the exploration phase should be guided, but cannot be performed by the designer because it requires too much time. In computer architecture research and development, simulation still represents the main tool to predict the performance of alternative architectural design points. If we consider a cycle-accurate system-level simulation, it requires a lot of simulation time and this time will continue to grow due to the increment of system complexities. For this reason, the exploration of the design alternatives can exceed the practical limits even if the number is not so large. This means that DSE requires being automatic and efficient in terms of optimization algorithms and predictive models, reducing the number of simulations to be done and thus saving time.

Moreover, novel HW/SW architectures continue to grow both in terms of number of interacting components and in terms of tunable parameters exposed to the designer. This makes the impact of novel parameter configurations difficult to predict, even when the designer knows the target applications and architecture. Furthermore, multiple competing objectives (e.g. Power and Performance) result in searching for multiobjective optima (Pareto solutions). In this context, the key challenges are to support the designer in a more systematic way through the design space exploration phase. 1st by decoupling the simulation infrastructure from the design space exploration environment with analysis support, 2nd to efficiently face the DSE search by using novel optimization strategies coupled with predictive models, and 3rd to combine exploration of hardware and software parameters considering both the design-time and run-time tuning of the target platform.

E. Integrated Framework

To the best of our knowledge, there is no integrated framework available that enables the estimation and simulative analysis of timing and power properties from a HW/SW independent executable specification model under manual mapping constraints, as described above. The goals of our presented approach are:

- Interface to model-driven SW design entry using MARTE/UML and the industry standard Matlab/Simulink model-based design environment,
- combination and augmentation of well-established commercial ESL synthesis and analysis tools into a seamless design flow enabling performance and power aware virtual prototyping from a combined HW/SW perspective,
- fast simulation and assessment of functional and extra-functional (time and power) properties of the entire system after platform mapping with scalable accuracy,
- framework for dynamic adaptation to changing context and transparent optimization of platform resource usage,
- multi-objective HW/SW co-exploration to assess the design quality and to optimize the system platform with respect to extra-functional properties.

III. PROPOSED CONCEPT

Our proposed concept for a rapid prototyping framework is illustrated in Fig. 1. As presented in Section II we follow the PBD approach with a separation of application ⟨3⟩, architecture ⟨3⟩, and mapping description ⟨3⟩. The architecture/platform consists of pre-existing IP components like processors, buses, hardware accelerators and memories, while the application describes how these resources are used to implement certain system functionality. For the specification of different domain-specific application workload scenarios, specified as use-cases ⟨3⟩ we propose to generate a system input stimuli specification ⟨4⟩ for triggering the executable system model.

The most important property of the proposed framework is that timing and power characterisation is separated from application specification and development. This separation allows platform providers to offer timing and power characterized virtual platform component models (IPs) ⟨3⟩. Together with the estimated custom HW ⟨3⟩ and SW components ⟨3⟩ a
timing and power aware executable virtual system prototype \( \mathcal{D} \) can be generated.

Based on the simulation trace \( \mathcal{D} \), obtained from executing the generated platform model, analysis tools \( \mathcal{D} \) can either generate a report or a visualization of the power consumption per system component over time \( \mathcal{D} \). The application of metrics on the trace is used to drive an automatic or semi-automatic exploration and optimization process \( \mathcal{D} \) that modifies different design parameters \( \mathcal{D} \) in a pre-defined design-space \( \mathcal{D} \). These parameters can be applied on the MDA design entry model, executable SystemC model, or the estimation and model generation tools.

The following sections give a more detailed description the different phases of our proposed rapid prototyping framework.

A. MDA Design Entry

The COMPLEX modeling entry is supported by the COMPLEX UML/MARTE modeling methodology that includes a toolset fully integrated in the Eclipse framework. This toolset automates the generation of the code which serves to generate the performable executable model. The UML/MARTE specification models both the system and the input stimuli environment.

The COMPLEX UML/MARTE methodology enables the specification of the different facets of the system in different model viewpoints. The COMPLEX model viewpoints are the Data Model View, the Functional View, the Communications and Concurrency (CC) View, the Platform View, the Architectural View, and the Verification View. These views enable separation of concerns and thus raise the level of abstraction as each view focuses on a specific aspect of interest of the system. The COMPLEX UML/MARTE methodology also defines the relationships among these views, and a workflow which guarantees them. This enables the designer to build a synthetic model (avoiding possible redundancies and thus coherence checks) and enables a cooperative workflow where the application and platform can be captured in parallel. The separation of concerns is given at several levels of the model design. The system (i.e., application mapped onto platform resources) is separated from the environment. Within the system model, the platform specification (i.e., processing resources, operating system) is separated from the model of the application. Finally, within the application, data structures, functionality (interfaces and classes) and application components are also separately captured. The extra-functional properties of the application are specified on the CC view, while the platform extra-functional properties are described in the Platform view. The Architectural view provides information about the allocation of application components onto the platform components and the DSE parameters and metrics can be reflected.

The COMPLEX UML/MARTE methodology follows also a component-based approach. At the application level, the designer packets the functionality within application components in the CC View. A system application is captured as a component, and instances of application components are used to capture the application architecture. This component represents the Platform Independent Model (PIM) according to the MDA paradigm. The platform architecture is captured in the Platform view by means of instances of SW and HW platform components, e.g., RTOS component instances, and instances of HW processor components. This architecture represents the Platform Description Model (PDM) according to the MDA paradigm. The component is the elemental unit used in the COMPLEX UML/MARTE methodology for deploying functions onto processing resources (i.e., microprocessors, FPGA) in the Architectural view. For instance, application components can be mapped onto the platform components which represent processing resources. Finally, the Verification view also uses component instances to reflect the connection between the system component and the environment components, it means, actors interacting with the system.

The COMPLEX UML/MARTE methodology has been explicitly designed for supporting design space exploration. Specifically, the methodology supports the specification of a design space, i.e., a set of design solutions, rather than a single design. The description of such design space is enabled by means of defining: a set of architectural mappings (allocation space), a range of values for platform attributes (parameters of the space), and several platform architectures (architecture space).

In order to specify this design space, the methodology relies on the MARTE profile and proposes new stereotypes for the missing semantics (i.e., DSE, IF-XACT concepts), which are proposed as a necessary enhancement of current capabilities of MARTE for embedded system design. Moreover, the COMPLEX UML/MARTE methodology supports also explicit constraints and rules that can be used by the designers to limit the space of solutions to those that are of main interest. The methodology also supports the specification of system local metrics. In contrast to global system metrics, such as total
power consumption, system dependent metrics depend on each specific system model. For example, the latency metric for servicing a specific function of an application component or the miss rate of the instruction cache of a given processor of the platform. This feature provides a capability of paramount importance: all the aspects of the system with some impact on its final performance: application and platform, SW and HW, architectures and architectural mappings, component attributes and different types of metrics, form now part of the DSE loop and therefore can be optimized at once, under a real holistic approach.

Finally, the COMPLEX UML/MARTE methodology allows designers to specify the input stimuli in a separated view, the Verification view. As mentioned, this view supports the specification of a set of environmental components and how they connect with the system component. This view provides a description of the interactions (through sequence diagrams) among the environmental components and the system component as the sequence of ordered messages in the context of a use case scenario. Timing information and ordering constraints among environment events are captured in the sequence diagrams, so that it enables the documentation and generation of realistic use cases. This is crucial for the dynamic performance estimation enabled by the executable model derived from the UML/MARTE system model. The methodology enables the definition of multiple scenarios that represent different use cases of the system. Designers may choose any scenario from those modeled in the Verification view to generate the performance executable model and explore the design space.

B. Executable Specification

The output of the MDA entry is an executable model generated from the PIM. The PDM is used to generate a structural platform model with virtual processing, memory, and communication elements. These are used to model the resource constraints of the execution platform. The generated executable specification (Algorithm Domain), platform description model (Architecture Domain), and platform mapping is shown in Fig. 2.

![Fig. 2. Example of an Executable Specification and a Platform Mapping](image)

a) Executable application model: In our executable application description model we perform a separation of behavior (computation) and protocol (communication). Our concurrent building blocks are tasks or processes that contain a behavioral and a protocol part. The behavior part describes the function or algorithm to be executed, written in sequential C/C++ code. This description is independent from an implementation in either HW or SW. Behaviors can be composed of functions and describe a pure sequential execution order. This enables reuse of existing software descriptions. Moreover, the tools mentioned in Section II-A allow synthesis to a C/C++ representation. An abstract task describes a runnable i.e. a process. Each abstract task contains a single behavior. Abstract tasks can either be active or passive. An active task starts running immediately after it’s activation and can either be blocked through a communication request or when its computation is finished. An active task can be (self) triggered again after a certain amount of time (time triggered task, periodic task). Passive tasks can only be triggered by active tasks through explicit requests. A passive task cannot trigger itself and it cannot trigger any other passive task.

The protocol part describes communication among behaviors. It is realized through a protocol that allows active tasks to call service on another passive task’s behavior. These tasks are blocking, i.e. the caller’s behavior can be continued after the service call has been completed. When multiple active tasks are requesting a service call of the same passive task a scheduling action is required. More details about this can be found in [6], [7]. These service calls abstract from a certain communication protocol implementation in either HW or SW.

b) System input stimuli: In order to examine and analyze the parallel application description model under a certain workload scenario, it needs to be simulated accordingly. The system stimuli might originate from user interaction or communication with other components that are part of the system’s environment. These stimuli describe use-case scenarios and can be derived from a UML/MARTE use-case specification or from an environment model in Matlab/Simulink.

c) User constrained HW/SW separation & mapping: The user-constrained HW/SW separation and mapping defines the binding of tasks from the application model to execution resources of the architecture/platform description model. Active and passive tasks can be mapped to execution resources, while passive tasks can only be mapped to memories.

d) Architecture/Platform description: The platform description model is composed independently from the application model. It is a pure structural and extra-executable representation of the execution platform consisting of execution resources (like SW processors, DSPs or ASICs), memories, communication resources (like shared buses), and pre-existing IP components. In addition, constraints that have a direct influence on the timing and power consumption are represented in the component’s meta-description. For SW processors it is the instruction set architecture (ISA) including its pipeline behavior, power modes, data and instruction cache models, and bus interfaces. For custom HW the used RT component
library, and for communication resources scheduling policies for shared media have to be specified.

C. Estimation & Model Generation

In order to allow a fast simulation and estimation, we create annotated C/C++/SystemC code. This annotated code contains information about the timing and power of each component. Power and timing information for each of them is obtained using existing and sophisticated tools as mentioned in Section II-B. As depicted in Section II a realistic system consists of components of different type e.g., custom hard- and software as well as IP-components, like communication infrastructure. Each component is estimated individually, using an appropriate tool. Based on the estimation an augmented version of the component is created, containing a power and timing model of the component. From these annotated components a virtual prototype is generated. This prototype is used to estimate the power and timing of the overall system. In the next four paragraphs we describe these steps in more detail.

e) Hardware/Software task separation: Depending on the user-defined mapping, each behavior of the parallel application description is estimated with an appropriate technique. The tools, used for HW & SW estimation and characterization perform a simulation-based estimation. Thus, each component must be simulated and characterized individually. The system is split into individual components. The surrounding system of the component serves as testbench/test environment during the simulation. This way we can simulate each component individually and still obtain estimates, which correspond to the behavior of the overall system.

f) Custom Hardware estimation: To consider the challenges of ASIC power-modeling mentioned in Section II we combine synthesis with cycle-accurate simulation at RT-level and a subsequent phase of basic-block identification and power/timing annotation. Although extensive power estimation at RT-level is very time consuming and thus not applicable in HW/SW co-simulation, it can be used as characterization approach for higher level estimation. This is why we apply lower-level estimation provided by the OFFIS PowerOpt tool to a small but typical testbench and derive cycle-averaged power estimates. These power values will then be further abstracted to basic block level and annotated to the internal control- and dataflow graph-representation (CDFG, see Fig. 3(a)). We differentiate between dynamic and static power as well as its source (e.g., functional units, controller, or clock tree). In addition the delay is annotated to each basic block which is fixed and can be statically derived from the cycle count within the scheduled CDFG and the frequency. In the last step we export the CDFG to a timing and power annotated SystemC module. The overall HW-estimation flow is presented in Fig. 3(a).

g) Software estimation: At the intermediate level of the COMPLEX tool chain, and integrated with software generation, hardware model characterization and design space exploration, the detailed software estimation toolset SWAT (Software Analysis Toolset) plays a key role. Its main goal is to provide more accurate and detailed performance and energy estimates with respect to those obtained using higher-level and more abstract models.

The purpose of SWAT is to provide methodology and tools to estimate performance and energy requirements of embedded software, conveying the advantages of ISS approaches and dynamic approaches, and mitigating their disadvantages. The SWAT approach is based on a statistical analysis and characterization of an intermediate assembly-level code representation, instrumentation and execution on the host machine. The most significant novelties of the proposed methodology are:

- The methodology does not require to execute an ISS, even in those cases where the software is required to interface to the external world, e.g. to hardware peripherals on the target SoC. Suitably designed stubs can be developed and seamlessly integrated with power FSMs models.
- The performance and energy consumption models of the target microprocessor can be derived in almost completely automatic way by means of a specific set of tools that are part of SWAT. The accuracy of such models depends on the accuracy of the basic figures available for the target core.
- The intermediate code (LLVM) is very close to the target assembly code, but still sufficiently abstract to allow for a source-level analysis. The analysis performed on the intermediate code, thus, combines efficiency, accuracy and flexibility.
- Execution on the host machine does not require the entire hardware of the target system, reducing costs and times of the analysis.

The SWAT modeling approach can be split into three main phases:

1) Target machine modeling Timing and energy consumption characteristics of the target architecture need to be modeled to in suitable mathematical form to feed the remaining phases of the flow. The first step, thus, consists in building this instruction-set model. This is
done only once for each target platform.

2) **Source code modeling.** The second modeling phase is aimed at decoupling the influence of the static structure of the source code (i.e., its semantics) both from the specific target architecture and, most notably, from the specific input run-time data.

3) **Dynamic behavior modeling.** The dynamic behavior is accounted for by performing a basic-block profiling at intermediate-representation level (i.e., LLVM code). Such a profiling can be accomplished by executing the code on a host machine, possibly resorting to some support libraries that model hardware devices.

4) **Post-processing.** A final phase combines all the models and derives timing and energy estimates of a specific (or a set of) execution. Beyond a wide range of statistics concerning static and dynamic properties of the application, this last phase back-annotates performance figures directly on the original source code. This is especially valuable to support the application developer in the subsequent optimization phase.

Each phase has been studied accurately and sound mathematical models have been built as a foundation. A simplified view of the resulting estimation flow is depicted in Figure 3(b).

**h) Pre-existing IP & virtual component models:** Despite custom HW and SW pre-existing or third-party components must be considered. Models for communication infrastructure like buses are provided by different vendors and typically are provided as parameterizable soft-macros, allowing an adaption to the system to be build. The macros already contain timing information but typically no information about power. Thus, communication power is estimated based on the TLM2 transport calls. Calculation accounts for the size of transferred data, type of access, as well as duration of the communication. Interruptions and re-arbitration events are also considered and must be delivered by the communication model.

General IP components delivered by third-party vendors cannot be estimated like custom HW and SW components. System-level representatives of these IP’s are typically provided as black-box executable models (e.g., API to a compiled object-file). These black-box modules typically contain timing but no power information. In order to obtain at least approximated power values a simple wrapper or monitor is used, which monitors the components in- and output. This information is used to control a power state machine (PSM) [8] inside the monitor. Power states and power values of the PSM are obtained from the component’s data-sheet.

Also the TLM architecture templates (initiator, target, memory wrapper, and router) used during virtual system generation are considered in this part of the flow. Together with the third-party components they are referred as virtual components. Each virtual component represents one design block of the architecture/platform description (Q). During virtual platform generation the estimated custom HW & SW behavior is integrated in these architecture wrappers to build an executable communication platform (see Fig. 2).

**i) Virtual system generation:** During generation of the virtual system annotated source from (Q) as well as the selected models from (Q) are combined to a virtual prototype. The example in Fig. 3(c) shows the virtual platform model obtained from the mapping specified in Fig. 2. The timing and power annotated execution-models are integrated with timing and power characterized platform models. In the example in Fig. 3(c) these platform models are: a TLM router with bus protocol and power model, and a system memory model. For the integration of the annotated task behavior with the TLM communication network we provide communication interface (IF) templates. These interfaces translate the function calls of the active tasks into TLM transaction containers. For passive tasks we synthesize a memory interface which decouples the TLM transactions from the activation of the behavior. That means the transaction is stored in the memory completely, before the passive task is activated. These interface templates are timing and power characterized for the chosen platform.

**D. Simulation**

**j) Pre-optimized power controller:** The pre-optimized power controller implements a framework for dynamic adaptation to changing context and transparent optimization of platform resource usage. It follows a distributed and hierarchical approach. On the one hand, a Global Resource Manager (GRM) is loaded on the host processor of the platform. It is a software task running in parallel with the application. It is a middleware providing a bridge between the application, the user and the platform. It conforms to practices of each Local Resource Manager (LRM) in each platform IP core (e.g., HW block or SW processor). It is used to adapt both platform and application at run time and to find global and optimal trade-offs in application mapping based on a given optimization goal. On the other hand, each IP core can execute its own resource management without any restriction, through an LRM. Such an LRM encapsulates the local policies and mechanisms used to initiate, monitor and control computation on its IP core.

In contrast to the collaboration between the GRM and the LRM, the collaboration between application and user is visible to the application developer and is performed as follows. First, the QoS requirements and the optimization goal are set by the user. The goal is then translated into an abstract and mathematical function, called utility function (e.g., performance, power consumption, battery life, QoS weighted combination of them). Then, at run time, the GRM manages and optimizes the application mapping taking into account the possible application configurations explored at design time. The platform resources currently available, the QoS requirements, and the utility function.

**k) Timing & power aware executable virtual system prototype in SystemC:** During system execution the annotated timing and power information is collected. Depending on the workload model different execution paths, leading to different timing and power values, are possible. After simulation, the collected information can be illustrated in a power-over-time.
diagram or can be used for a power-breakdown. Our annotations can be traced at different levels of granularity to allow a user-defined trade-off between performance and accuracy. Fig. 3(d) depicts the different timing and power evaluation levels.

On the most abstract level we consider analysis on task granularity. This can be easily performed between active and passive tasks with blocking communication relation. Execution time and power of the passive task can simply be isolated and accumulated with time and power of the active task. The next level of granularity works on communication granularity. Power and timing of computation nodes in the communication graph are accumulated and only traced at the time points of communication. For a deeper analysis of the timing and power behavior traces on basic block granularity of a CDFG is also possible.

A possible new simulation view provided by the COMPLEX project is the so-called Network View enabled by the SystemC Network Simulation Library (SCNSL) [9]. It is an extension of SystemC to allow modeling packet-based networks such as wireless networks, Ethernet, and field bus. It supports the simulation of packet transmission, reception, contention on the channel and wireless path loss. The traditional virtual platform, made of CPU, HW blocks and bus, can be extended by wrapping it into a network node exchanging packets with other nodes through a channel by using well-known protocols such as IEEE 802.15.4. All these elements are provided by SCNSL as traditional SystemC blocks. SCNSL also allows improving timing and power analysis by introducing the effect of communications which have a direct impact on timing behavior and power consumption of the system under design.

E. Exploration & Optimization

The Exploration and Optimization phase is part of the design flow capable to create a feedback loop between the performance estimation part (including MDA Design Entry, Executable Specification, Estimation & Model Generation and Simulation phases) and the parameters configuration of the target system. In particular, the loop is closed by acquiring the system simulation traces that need to be post processed before being presented either to the user or to an automatic framework for exploration and optimization. Then, according to the information gathered by the previous simulation, a new HW/SW system configuration will be selected within the design space. In the next paragraph we describe these steps in more detail.

1) Simulation trace: Our proposed simulation and tracing framework supports different granularities for each task. Moreover, it is possible to define certain regions of interest in the sequential behavior of a task that can be investigated down to basic block granularity.

m) Trace analysis: The simulation and analysis of timing and power traces allows an evaluation of the chosen application mapping, the performance of the architecture and the effects of the synthesis constraints. Different design configurations or iterations with adjusted mapping, platform composition and constraints allow multi-objective design-space exploration.

n) Visualization & reporting: The visualization and reporting step in the proposed framework has in charge the presentation of the results coming from the trace analysis phase in a readable format for the designer. This phase is necessary to have the designer driving the exploration loop. Moreover, this phase includes also the visualization of the
analysis coming from the exploration loop done by using the MOST tool.

\textit{o) Exploration & optimization}: Starting from the definition of the design space, the exploration and optimization step iteratively generates an instance of the design space based on the knowledge acquired by the post-processing of the simulation traces of previous selected configurations. The exploration phase is a step in the design flow that is needed for surfacing the design space (changing the system parameters) in order to find the optimal system configurations among all the possible alternatives that are part of the design space. Moreover, the design space exploration loop is also used to determine some knowledge about the system parameters (such as the main effects, interaction effects) and design space (such as, configuration distribution with respect to the system performance). This phase can be done by using a user-centric DSE or an automatic DSE phase. The goal in using an automatic design space exploration and optimization tool is in the fact that it should be able to automatically interact with system models in order to avoid the intervention of the designer for the DSE phase (except for the analysis of the results) once the target problem is formally defined. The automatic design space exploration tool used within the project is called MOST, and the interfaces used to interact with the other components of the design flow are those presented in [10]. On the other side, the usage of a user-centric DSE flow is needed when a detailed analysis of the system behavior is necessary (e.g. trace analysis or time behavior), once the problem cannot be formally defined or it is not easy to be defined, or when the automatic modification of the parameters on the system model is not possible or requires a larger modeling effort.

\textit{p) Design space definition}: The design space is defined by the list of tunable parameters available on the HW/SW platform. Moreover, it includes the set of possible parameters and the rules defining some cuts within the design space eventually due to interferences between parameters. The design space definition represents the degrees of freedom that the designer or an automatic tool can have for tuning the HW/SW platform.

\textit{q) Design space instance parameters}: A design space instance is a valid configuration of parameters within the design space defined before selected or by the designer or by an automatic tool. It is composed of a value for each parameter of the design space, ranging within the set of available levels. Those values will be used to fill the right parameter values in several stages of the design-flow (see Figure 1). In the project, the list of parameters ranges at the MDA Design Entry and Executable Specification levels from functional reimplementation to mapping of HW/SW tasks and IP selection, while at Estimation and Model Generation level from IP and memories configuration to selection of embedded software optimizations and run-time management strategies.

\section*{IV. Conclusion}

In this paper we presented the COMPLEX reference framework and design flow allowing rapid virtual prototyping of heterogeneous embedded HW/SW systems under consideration of timing and power aspects. The presented approach enables a UML/MARTE design entry with automatic generation of an executable SystemC model. The extra-functional estimation flow considers custom hard- and software as well as third party IP components. For each of these different types we use state-of-the-art commercial tools, inducing satisfying estimates for individual components. A generated virtual prototype allows a fast and unitary simulation of the complete system’s functional and extra-functional properties, which helps the designer to identify and eliminate performance bottlenecks and power peaks. Thus, a fast and early trade-off between different design alternatives becomes feasible. To support an efficient exploration we have extended our flow by a feedback loop, allowing guided iterations through different configurations, mappings, power management policies etc. We are convinced that the overall framework will integrate existing approaches and point-tools, into a common and unified flow for timing and power aware rapid prototyping.

\section*{Acknowledgment}

The authors would like to thank all partners involved in the COMPLEX FP7 European Integrated Project [11], funded by the European Commission under Grant Agreement 247999.

\begin{thebibliography}{99}
\item The COMPLEX project (247999), "Codeign and power management in platform-based design space exploration," http://complex.ofis.de.
\end{thebibliography}