SOCROCKET: A VIRTUAL PLATFORM FOR SOC DESIGN

Luca Fossati[1], Thomas Schuster[2], Rolf Meyer[2], and Mladen Berekovic[2]

[1]*European Space Agency (Keplerlaan 1, 2201AZ, Noordwijk, NL - Tel. +31 71 565 8540) - Luca.Fossati@esa.int*
[2]*Technische Universitat Braunschweig (Muhlenpfordtstr 23, Braunschweig, DE) - schuster,meyer,berekovic@c3e.cs.tu-bs.de*

1. INTRODUCTION

Both in the commercial and in the aerospace domain, the continuous increase of transistor density on a single die is leading towards the production of more and more complex systems on a single chip, with an increasing number of components. This brought to the introduction of the System-On-Chip (SoC) architecture, that integrates on a single circuit all the elements of a full system. This strive for efficient utilization of the available silicon has triggered several paradigm shifts in system design. Similarly to what happened in the early 1990s, when VHDL and Verilog took over from schematic design, today SystemC and Transaction Level Modeling [1] are about to further raise the design abstraction level. Such descriptions have to be accurate enough to describe the entire system throughout the phases of its development, and has to provide enough flexibility to be refined iteratively up to the point where the actual device can be produced using current process technology. Besides requiring new languages and methodologies, the complexity of current and future SoCs (SCOC3 [16] and NGMP [5] are example in the space domain) forces the SoC design process to rely on pre-designed or third party components. Components obtained from different providers, and even those designed by different teams of the same company, may be heterogeneous on several aspects: design domains, interfaces, abstraction levels, granularity, etc. Therefore, component integration is required at system level. Only by applying design re-use it is possible to successfully and timely design such complex SoCs.

This transition to new languages and design methods is also motivated by the implementation with software of an increasing amount of system functionalities. Hence the need for methodologies to enable early software development and which allow the analysis of the performance of the combined Hw/Sw system, as their design and configuration cannot be performed separately. Virtual Prototyping is a key approach in this sense, enabling embedded software developers to start development earlier in the system design cycle, and cutting the dependency on the physical system hardware.

In order to successfully implement the described methodologies, it is requested to have access to the a wide selection of IP-Cores (and related SystemC/TLM models) and access to the latest Electronic Design Automation (EDA, [17]) tools. On the one hand, for what concerns the European Space landscape, such IP-Cores are provided by the European Space Agency [4] and a few other suppliers (e.g Aeroflex Gaisler with GR-LIB [2]). On the other hand, for what concerns the related high abstraction models and related design methodologies (partly depicted in Figure 1), the European Space Agency, through the Braunschweig Technische Universitat, has started the development of the **SoCRocket Virtual Platform** [8]. Together with the Virtual Platform infrastructure SoCRocket contains a library of IP-Core models. The SoCRocket library has been built around the TrapGen LEON instruction set simulator [15]. The library contains a variety of SystemC simulation models such as caches, memory management unit, AMBA interconnect, memory controller, memories, interrupt controller, timer and more. All models are TLM2.0 compliant and come in both loosely-timed and approximately-timed coding styles. As later-on presented more in detail, the runtime reconfiguration, the completeness of tools and models, as well as the fact that all simulation IPs have a freely available RTL counterpart differentiates SoCRocket from other commercially available Virtual Platforms. Moreover, due to their TLM2.0 compliance the provided models are not bound to the SoCRocket environment but they can be used with alternative tools,

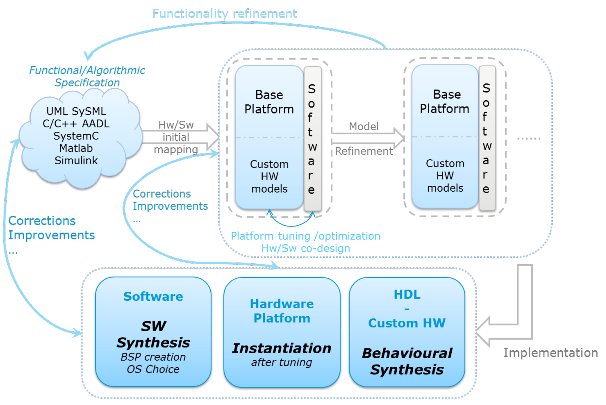

Figure 1. Part of the steps composing a modern Electronic System-Level design flow.

such as Cadence Virtual Platform [3] or Synopsys Platform Architect [10].

The paper is organized as follows: Section 2 presents the architecture of SoCRocket and the related library of SystemC models. Finally Section 3 shows how SoCRocket was used to optimize the design of a LEON3-based SoC targeted to the execution of an implementation of the CCSDS standard n.123 for the lossless compression of hyperspectral images.

2. SYSTEM DESIGN WITH SOCROCKET

As highlighted in Section 1 *Virtual System Prototyping* brings added value to multiple parties, in the first place to system software developers, by enabling a true early start of software design and by promoting the concurrent engineering practices for software and hardware development. Also, it avoids time-consuming hardware-software integration late in the development cycle as such integration can be performed and verified on the virtual prototype. In addition, a virtual platform has a number of unique advantages over its physical counterpart. It is cost-effective compared to physical solutions, easier to distribute and deploy as it has no wires and pins. Most enjoyed by developers, a virtual prototype allows unlimited observability and controllability of the target hardware, not limited to the available pins on the prototype.

SoCRocket combines a Virtual Platform based on the SystemC and TLM standards and a library of component models to be combined into the SoC to be simulated.

Before introducing the Models Library, this Section focuses on the SystemC and TLM standards used to structure the library itself; later-on SoCRocket capabilities are described.

2.1. SystemC and TLM

As stated in [12], "SystemC is a system design language that has evolved in response to a need for a language that improves overall productivity for designers of electronic systems". SystemC [11] is a C++ based modeling platform supporting design abstractions at the register-transfer, behavioral, and system levels. It consists of a set of C++ libraries devoted to building system-level executable models of mixed hardware-software systems; many constructs are provided, among which modules, channels, and interfaces. It allows the creation of executable simulators at several levels of abstraction. SystemC encourages the designer to concentrate on the functionality rather than on the actual HW structure, offering consistent productivity gains in the initial design phases.

Using a language for System-Level modeling, such as SystemC, is not enough for building an effective simulatable model: it is also necessary to define modeling styles and interoperability rules among the various models. Transaction Level Modeling (TLM), first introduced in [13], is now widely accepted as an efficient technique for abstract modeling of communication and computation. In addition to standardizing communication among models, TLM separately addresses communication and computation enabling the designer to face the two issues in separate moments.

Table 1 shows the complexity of SystemC, TLM models compared to the RTL counterpart; note that the SystemC column includes both the LT and AT coding styles.

2.2. Models Library

The *SoCRocket Models Library* consists of a collection of IP-Cores described with the C++ language and adhering to the SystemC and TLM IEEE standards; currently only part of the Aeroflex Gaisler GRLib is modeled, but the library is easily extendable with any other model compliant to the SoCRocket modeling guidelines. Figure 2 shows the IP-Cores currently modeled. All of them are fully parametrizable to be able to simulate all the possible configurations allowed by the corresponding RTL IP-Core and even further, to enable exploration of alternative hardware configurations.

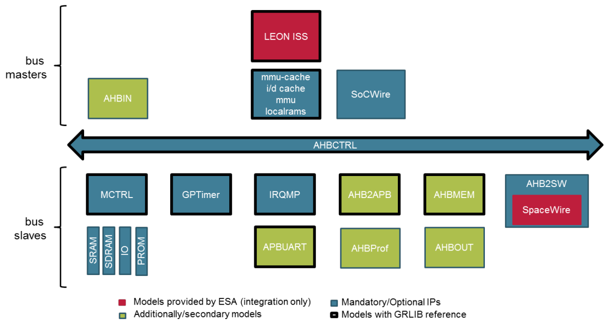

Figure 2. SoCRocket Models Library

The SystemC and TLM methodologies allow the use of various abstraction levels and coding styles; SoCRocket library focuses mostly on what is described in the Accellera TLM2 manual [18] as Loosely Timed (LT) and Approximately Timed (AT) coding styles. The LT configuration of the simulation models is intended for fast address-accurate simulation (SW development). Communication is modeled using blocking function calls and as little synchronization with the SystemC kernel as possible. The AT configuration of the simulation models is intended for architecture exploration. To provide the therefore required accuracy it models selected features of the involved communication protocols. This mainly relates to the pipelined nature of AMBA AHB bus. the AT abstraction of the IPs in this library do not model the AMBA AHB protocol in a cycle-accurate way; this because the AT mode is supposed to provide a reasonable

Model	SystemC - LT+AT	VHDL
AHCTRL	868	1675
APBCRTL	329	1183
MMU_CACHE	3136	6639
GPTimer	540	209
APBUart	245	482
MCTRL	937	1829
IRQMP	432	262

Table 1. Complexity of the SystemC models with respect to their RTL counterparts

speedup over RTL simulation, while still being accurate enough to facilitate architectural decisions.

Given its flexibility and configurability, this platform is an ideal starting point to design and shape new SoC architectures or to optimize existing ones. It helps to identify risks and flaws in early stages and allows early software development and simulation as close to the real-thing as possible.

2.3. Design Flow

In addition to appropriate tools (SoCRocket in this case) it is important to set-up a methodology guiding the designer in their use. The SoCRocket-based design flow (Figure 3) starts from the software implementing the desired functionality. In the first step of the flow the reference software is manually partitioned into elements to be implemented as hardware accelerators and as software tasks running on the system's processor(s). Finding the right partitioning for a system is a complex task and usually requires multiple iterations; in this area SoCRocket offers great advantages by enabling to easily map C/C++ code into SystemC components (hardware modules) or software tasks. While the custom hardware models can either be manually created or generated using high-level synthesis tools, the base hardware platform is modeled by assembling the simulation IPs contained in the Model Library and, currently, representing the core components of the GRLIB hardware library (see Figure 2). The second,

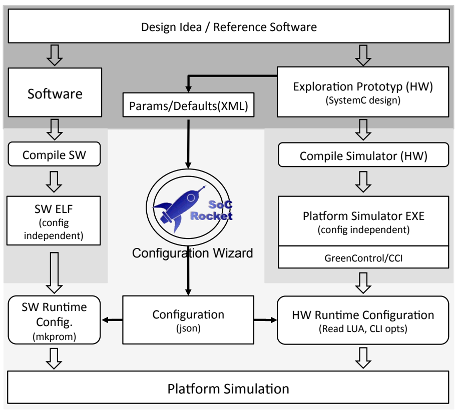

Figure 3. SoCRocket-based design flow

and equally important, step consists of the identification of an optimal configuration of the various hardware elements, by appropriately tuning their parameters, such as the cache size, associativity, but width, etc. even the number of processing units.

Note how, often, those two design phases are not separately executed, but they take place at the same time, iteratively changing the Hw/Sw partitioning and the system configuration until a satisfactory system is obtained. SoCRocket provides support during the whole design flow by easily enabling, as shown later-on in Section 3, changes in the models' configuration parameters and by seamlessly allowing the move of functionalities between hardware and software. Moreover, as explained in Section 2.4.1, appropriate tool and mechanisms are provided to enable collecting and analyzing performance measures and statistics about the simulation, helping the designer in the assessment of the quality of the architecture. If it is not satisfactory the system parameters are changed and another exploration executed.

As demonstrated in section 3.2 it is possible to automatically drive such design space exploration by a small shell script, which systematically adapts the configuration and extracts simulations results.

2.4. Virtual Platform Infrastructure

So far the Model Library and the Design Flow have been presented; having a collection of models is not enough without the appropriate facilities to interconnect them, manage the simulation (start, stop, pause, etc.) and, especially without the ability to produce results and statistics which allow us to judge the quality and the performance of the simulated system and which guide the designer in the optimization of such system.

SoCRocket features a graphical user interface, called *Configuration Wizard*, CW, which simplifies the work of the user in instantiating the various SystemC models and in setting the values for their configuration parameters. The outputs of the CW are the SW runtime configuration and the HW runtime configuration. Next to defining parameters, the CW interprets the system's memory map, generates linker scripts for software mapping, and creates a compilation script for automatic integration of the new system in the platform's build environment. Running platform simulation still requires mapping the software portions of the design. The predefined SW stack generated by the CW can be used either for bare-metal simulation or for being integrated in the RTEMS Operating System [7].

The next paragraph explains what SoCRocket offers in terms of facilities for recording and analyzing statistics about the simulation being run.

2.4.1. Performance Monitoring

During simulation the platform infrastructure gathers execution statistics. For this purpose every simulation model provides a set of performance counters. In the default configuration the performance counters are written to the terminal after the end of the simulation. It is also possible to trace counters in log or waveform files, and to register handles for certain bounds and events; The latter is achieved relying on GreenSocs infrastructure [6]. All the execution statistics support the assessment of the design goals, usually throughput and latency at the lowest possible cost (typically, silicon area or power consumption).

All performance counters in the system can also be accessed at any time during simulation using the build-in analysis API. The analysis API allows listing parameters, to read/write/display parameters and to log parameters. Furthermore, it is possible to create waveforms for parameters and to create callback functions for custom analysis tools.

Simulated Time Analysis The facilities provided by the SystemC kernel are used for keeping track of simulated time. Each model is annotated with the time duration of its activities and events (for example the duration of each assembly instruction in a microprocessor) and when such events occur the global system time is updated accordingly. As the timekeeping infrastructure is directly part of the SystemC kernel, the complex part in order to achieve accurate time results consists in determining the time duration of the various events; Section 3.1.1 presents the timing accuracy of the Model Library using the VHDL-RTL models as reference.

Power Analysis Power modeling consists of equipping the IP models with capabilities for simulating not only the behavior of the target hardware component, but also their energy consumption in the different operating conditions. Achieving this is more complex than measuring simulated time, as no facilities are provided by the SystemC kernel, so they have to be ad-hoc implemented. The SoCRocket library provides an event based power-monitoring system. The basic assumption behind the implemented mechanism is that the power consumption of each component depends only on the component's own behavior, since the power estimation framework is built over an architectural simulator, already taking into account the interaction between components.

Three different classes of power dissipation are supported: static power, dynamic internal power, and dynamic switching power. Each of them is estimated and reported separately. The static power represents the 'leakage' of the component. Therefore, SoCRocket simulation models contain at least one input parameter, which is supposed to be initialized with normalized leakage power information. The dynamic power of a component is composed of an internal power portion and a switching power portion. In general, dynamic power is linearly dependent on the clock frequency. The application dependent part of the dynamic power is the so-called switching power. Most library components, such as memories and busses, assign energy quotas to read/write operations, while e.g. the processor uses a fixed energy per instruction budget.

To estimate the power consumption of the various models, all relevant GRLIB components have been synthetized using a generic design kit for 90nm CMOS technology [9]. Of course this is just to provide a usage example, the user will have to repeat the characterization of the components for each target technology it intends to use.

3. EXPERIMENTAL RESULTS

This Section presents a quantitative assessment of the platform's capabilities and performance. In particular we are interested in how well the models reflect the respective hardware components: while 100% accuracy cannot be achieved (otherwise they would be the component themselves) it is important that they are accurate enough for the purposes for which they are to be used. As explained in Section 2.4.1, next to functional behavior we are interested in how well the models approximate the execution time and the power consumption of the hardware elements they model.

The last part of the section is devoted to presenting an example of how SoCRocket can be used to assemble and configure a LEON3-based System-on-Chip targeted to the execution of a lossless compression algorithm for hyperspectral images. Results show that the design space can be extensively covered in a short time and that the produced statistics consistently help in determining an optimal configuration.

3.1. Virtual Platform Capabilities

When considering hardware components there are various metrics which are of interest to the designer; they include, among others, area occupation, power consumption and, of course, execution speed. SoCRocket library, as explained in Section 2 currently focuses on execution speed and power consumption. In order to be able to provide meaningful results it is important to know how accurate the models reflect the real hardware.

3.1.1. Models' Accuracy

Figure 4 shows the overall accuracy of the simulation of a LEON3-based SoC when executing various benchmarks; both loosely-timed and approximately-timed coding styles are used, and the results compared with VHDL simulations. While the accuracy is not 100% (as expected, in a model some details have to abstracted) it is sufficiently high to provide meaningful results. Moreover, and most important, the measurements follow the same trend than the RTL counterpart: when, as a consequence of hardware changes (e.g. doubling of cache size) the execution time decreases, it also decreases in the simulation. Such a property is fundamental to properly carry out design space exploration to optimize the system under analysis.

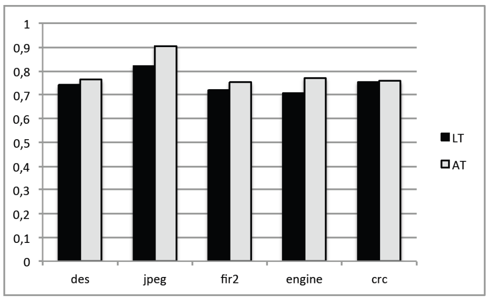

Figure 4. Timing accuracy of LT and AT simulations with respect RTL

Validation of the power estimation framework has not been completed yet.

3.1.2. Performance Results

Table 2 shows the speedup of the virtual platform with respect to the RTL description of the system on the same benchmarks on which the accuracy was measured. Indeed the less than 100% timing accuracy is balanced by the huge execution speed-up, more than 3 orders of magnitude over RTL.

Model	**LT**	**AT**	**VHDL**	**SpeedUp**
des	6.1	10.33	8707	1427
jpeg	6.51	11.26	7460	1145
fir2	83	140.87	110115	1329
engine	76	133.62	109143	1441
crc	15	25.18	19489	1272

Table 2. Simulation performance (in sec.); SpeedUp is measured for the LT model over VHDL

3.2. Usage Example

The general design flow is depicted in Figure 3. In the first step of the flow, the reference software is segmented into two parts: one that will become the software running on one of the target processors, and one that will become the hardware; to simplify this example we assume that that no custom hardware accelerators are used.

After the partitioning, the base hardware system must be instantiated, either manually (by explicitly writing

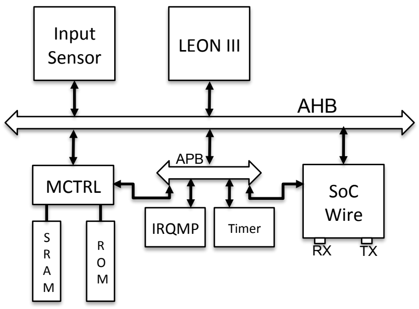

Figure 5. Hardware Platform targeted to the execution of the CCSDS Lossless compression algorith for hyperspectral images.

C++ code) or through the Configuration Wizard and software is compiled and mapped onto it. As explained in Section 2, during simulation, the platform infrastructure gathers execution statistics and the designer, by analyzing those, can decide the changes to apply to the hardware configuration. Simulations can then be re-executed to validate the changes. Scripts are also provided to automate those steps.

3.2.1. Design Flow Application to CCSDS Lossless Compression Algorithm

The application selected for this example consists of an implementation of the CCSDS Standard n. 123 for the lossless compression of hyperspectral images [14]. A hyperspectral image can be regarded as a stack of individual images of the same spatial scene, with each such image representing the scene viewed in a narrow portion of the electromagnetic spectrum. Overall, the algorithm is based on adaptive linear predictive compression using the sign algorithm for filter adaptation, with local mean estimation and subtraction. The prediction residual is then encoded using a sample-adaptive Golomb-Rice encoder. The code itself is in large parts dominated by data movements and bit-level optimizations.

A single-processor configuration with small 2-set associative instruction and data caches was selected as a starting point for design space exploration. Figure 6 presents the optimal tradeoffs obtained by varying the cache size and the number of sets; many more points of the design space were obtained during its exploration, but the ones presented in the diagrams are the ones on the Pareto curve, better than any other point for one of the metrics of interest (in this case execution speed and power consumption). The designer is then able to choose among those points the system configuration which better suits his needs.

Note that while this example provides an indication of the capabilities of SoCRocket, it is a simplification of the design process, as, for example, it does not consider the steps of hardware/software partitioning and it limits the design space to the cache configuration.

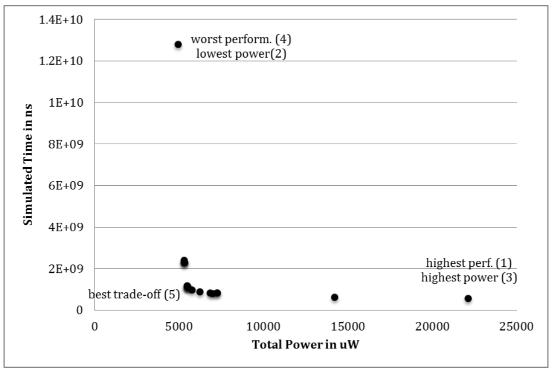

Figure 6. Optimal System configurations

REFERENCES

1. Accellera Systems Initiative: `http://www.accellera.org`.
2. Aeroflex Gasiler GRLIB: `http://www.gaisler.com/index.php/products/ipcores/soclibrary`.
3. Cadence Virtual Platform: `http://www.cadence.com/products/sd/virtual_system/pages/default.aspx`.
4. ESA IP-Cores: `http://www.esa.int/TEC/Microelectronics/SEMVWLV74TE_0.html`.
5. ESA Next Generation Microprocessor (NGMP): `http://microelectronics.esa.int/ngmp/ngmp.htm`.
6. GreenSocs: `http://www.greensocs.com/`.
7. RTEMS home page. http://www.rtems.com/.
8. SoCRocket Web Page: `http://www.esa.int/TEC/Microelectronics/SEMW9XK1QAH_0.html`.
9. Synopsys Generic 90nm Design Kit: `http://www.synopsys.com/Community/UniversityProgram/Pages/Library.aspx`.
10. Synopsys Platform Architect: `http://www.synopsys.com/Systems/ArchitectureDesign/Pages/PlatformArchitect.aspx`.
11. G. Arnout. SystemC standard. In *Asia and South Pacific Design Automation Conference, 2000*, 2000.
12. D. C. Black, J. Donovan, B. Bunton, and A. Keist. *SystemC: From the Ground Up*. Springer, 1 edition, May 2004.
13. L. Cai and D. Gajski. Transaction level modeling: an overview. In *CODES+ISSS*, 2003.
14. CCSDS. Lossless Multispectral & Hyperspectral Image Compression. 2012.
15. L. FOSSATI. TLM 2.0 Standard into Action: Designing Efficient Processor Simulators - http://www.design-reuse.com/articles/26286/tlm-2-0-processor-simulator.html. *Proc. of 19th IP based electronic system conference and exhibition (IP-SoC)*, 2010.
16. F. Koebel and J.-F. Coldefy. SCOC3: a space computer on a chip. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1345 –1348, March 2010.
17. D. MacMillen, M. Butts, R. Camposano, D. Hill, and T. W. Williams. An Industrial View of Electronic Design Automation. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, 19(12):1428–1448, 2000.
18. Open SystemC Initiative (OSCI). OSCI TLM2 USER MANUAL, Nov. 2007.