Virtual Platforms for Mixed Time-Criticality Applications: The CoMPSoC Architecture and SDF3 Design Flow

Benny Akesson[1], Sander Stuijk[1], Anca Molnos[2], Martijn Koedam[1], Radu Stefan[1], Andrew Nelson[2], Ashkan Beyranvand Nejad[2], and Kees Goossens[1]
[1] Eindhoven University of Technology, [2] Delft University of Technology

I. Introduction

Systems-on-Chip (SoC) complexity increases as a growing number of applications are integrated and executed on contemporary systems. These applications consist of communicating tasks mapped on heterogeneous multi-processor platforms with distributed memory hierarchies that strike a good balance between performance, cost, power consumption and flexibility [1], [2]. Complexity is further increased by an increasing number of *use-cases*, which are different combinations of concurrently running applications. The applications have *mixed time-criticality*, which is a mix between firm, soft, and no *real-time requirements*. Firm real-time requirements must always be satisfied to prevent unacceptable output quality loss, while occasional failures to meet soft requirements can be tolerated. Lastly, non-real-time applications do not have well-defined timing requirements, but must still be responsive.

Applications from different domains and that have different time criticalities use different *design and verification methods*. After applications are developed, the verification process begins. Verification of real-time requirements is traditionally performed using formal analysis, simulation, or a combination of the two. Firm real-time applications demand rigorous formal verification, since their requirements must always be met. In contrast, soft real-time applications are often verified by simulation for a large set of inputs, as they are often dynamic by nature and difficult to verify by formal methods in a cost-effective manner.

To reduce cost, platform resources, such as processors, interconnect, and memories, are shared between applications. However, resource sharing causes *interference* between applications, making their temporal behaviors inter-dependent. This results in three problems with respect to system design, verification, and integration. Firstly, accurate system-level simulation and several approaches to formal analysis in complex SoCs are infeasible, because of the *state-space explosion* resulting from the many use-cases, application inputs, and resources states. Secondly, use-case verification becomes a *circular process* that must be repeated if an application is added, removed, or modified [3]. Thirdly, it is difficult to support various automatic analysis and design flows. As a result, the integration and verification process is a dominant part of SoC development, both in terms of time and money [3].

The CoMPSoC platform [4] addresses these problems by executing each application in an independent virtual platform, and by using the SDF^3 design flow [5] that automatically analyses firm real-time applications and maps them on a virtual platform. The CoMPSoC virtualization technology relies on two complexity-reducing concepts: *composability* and *predictability*, detailed as follows.

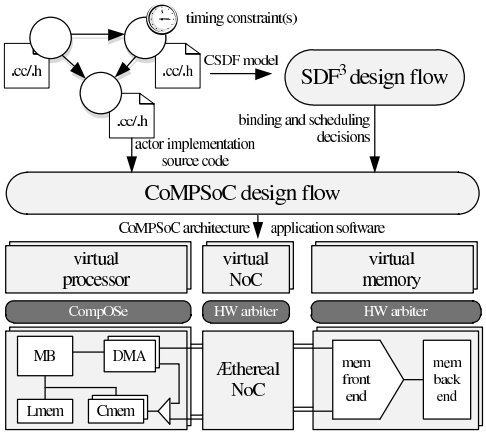

Fig. 1. CoMPSoC architecture and SDF^3 design flow.

Composable virtual platforms are completely isolated and cannot affect each other's temporal behaviors by even a single clock cycle. They are hence *virtualized in terms of actual execution time*, enabling applications to be designed, developed and verified in isolation. This alleviates the verification problem in the mixed time-criticality domain in three ways: 1) verification becomes a non-circular process, 2) the time required by simulation-based verification is reduced, since only a single application in its virtual platform has to be simulated, and 3) the use of different design and verification methods is enabled.

The virtual platforms are also predictable, which means that all platform and application interference is bounded. This makes them *virtualized in terms of performance bounds*, such as upper bounds on latency or lower bounds on throughput. This enables firm real-time applications to be verified using formal performance analysis frameworks, such as data-flow analysis [5]. Composability and predictability are hence complementary concepts that both solve important parts of the verification problem for mixed time-criticality systems, and provide a complete solution when combined.

II. Platform architecture

The CoMPSoC platform, illustrated in the bottom part of Figure 1, has a tiled architecture consisting of processor tiles, network-on-chip, and memory tiles. Each of these tiles and the interconnect are virtualized to implement a set of virtual platforms. We first briefly describe each resource in turn

and then present the techniques to achieve composable and predictable virtualization.

A *processor tile* is equipped with a MicroBlaze processor running the CompOSe real-time operating system [6]. CompOSe provides composable and predictable services, such as application scheduling and power management [7]. A processor tile furthermore contains non-shared local memory (Lmem) for instruction and data, as well as communication memories (Cmem) used by a DMA for communication with remote tiles. The *memory tile* is subdivided into a front-end and a back-end. The front-end is independent of memory technology and contains buffering and arbitration. The back-end interfaces with the actual memory device and is different for different types of memories. It is possible to use an off-the-shelf SRAM back-end, but a customized SDRAM back-end [8] is used to enable efficient performance virtualization. The tiles in the system are interconnected using the Æthereal network-on-chip [9]. The architecture of the network comprises network interfaces that packetizes and buffers incoming data, and control access to the network, and routers that forward packets towards the destination network interface.

The CoMPSoC platform uses three main techniques for composability and predictability, respectively. The techniques for composability are [4]: 1) use *preemption* after a fixed time to prevent large or infinite requests from one application from starving other applications, 2) *delay scheduling* until the end of a time slice to prevent requests that finish early from affecting when the following request is scheduled, and 3) use *composable scheduling*, such as time-division multiplexing (TDM), where the presence or absence of requests from one application cannot affect when other applications are scheduled. The techniques for predictability [4] are: 1) enable *worst-case analysis per resource* by requiring that all data for a request is available and that there is sufficient memory space to store responses before scheduling it, 2) use *predictable resources* with bounded worst-case execution times, such as the CoMPSoC processing tiles, network-on-chip, and memory tiles, and 3) use *predictable scheduling* to bound worst-case response times, such as TDM or Round-Robin.

III. Design flow

Programming heterogeneous systems, such as the CoMPSoC platform, is a very challenging task. Model-based design approaches using the dataflow Model-of-Computation have emerged as a promising solution to address this challenge. For example, [10] presents a design flow that maps a throughput constrained application, modeled with a scenario-aware dataflow graph, to an MPSoC. This design flow is implemented in the SDF^3 tool set [5]. We adapted this design flow for use with CoMPSoC. As shown in Figure 1, our flow takes an application modeled with a Cyclo-Static Dataflow (CSDF) graph [11] as an input. The nodes in a CSDF graph, called *actors*, model application tasks and the *edges* model control or data dependencies. In the CoMPSoC design flow, each actor is associated with C code that implements its functionality. When determining the mapping of the actors on the platform resources, the design flow abstracts from this functional behavior. To provide timing guarantees, it only needs to consider the worst-case execution time and worst-case memory requirements of an actor. Currently, a designer is responsible for providing these inputs to our flow.

To implement a CSDF graph, its actors and edges should be bound and scheduled on the resources of an MPSoC. This process is handled by our design flow. Our flow first analyzes the trade-off between the storage-space assigned to the edges and the throughput of the graph. After constraining the storage space of the edges, the flow binds the actors to the MPSoC resources. Next, static-order schedules are constructed for all processors to which actors have been bound. Finally, the flow computes the minimal TDM time slices needed on these processors to guarantee the timing requirements of the application. By minimizing the TDM time slices, processor resources are saved for other applications. Once the complete mapping is known, the CoMPSoC design flow generates a set of C source files that together with the C source code of the actors implement the complete application on the CoMPSoC platform. The generated C code contains all required function calls to CompOSe to initialize arbiters in the processor tiles, network-on-chip and memory tiles, and execute the application within its timing constraints.

IV. Case Study

A case study was performed in which an H.263 decoder, modeled with a CSDF graph, is automatically mapped to a two-tile CoMPSoC instance using our automated design flow. The flow was able to find a mapping of our application within seconds that satisfied its timing constraints as well as the resource constraints imposed by our platform instance. After running our flow, the CoMPSoC instance and our mapped application were implemented on an FPGA board using an automated synthesis trajectory. Experiments on the FPGA board confirmed that our platform provides a composable and predictable behavior when running the H.263 decoder.

V. Conclusions

This paper presents the CoMPSoC architecture, which combined with the SDF^3 design flow can be used to realize virtual platforms for mixed time-criticality applications. The CoMPSoC virtualization technology combines composability and predictability in a single platform and design trajectory.

References

[1] STMicroelectronics and CEA, "Platform 2012: A Many-core programmable accelerator for Ultra-Efficient Embedded Computing in Nanometer Technology," 2010, white paper.

[2] C. van Berkel, "Multi-core for Mobile Phones," in *Proc. DATE*, 2009.

[3] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. IEEE*, vol. 91, no. 1, 2003.

[4] B. Akesson *et al.*, "Composability and predictability for independent application development, verification, and execution," in *Multiprocessor System-on-Chip — Hardware Design and Tool Integration*, M. Hübner and J. Becker, Eds. Springer, 2010, ch. 2.

[5] S. Stuijk *et al.*, "SDF^3: SDF For Free," in *Proc. ACSD*, 2006.

[6] A. Hansson *et al.*, "Design and Implementation of an Operating System for Composable Processor Sharing," *MICPRO*, vol. 35, no. 2, 2011.

[7] A. Nelson *et al.*, "Composable power management with energy and power budgets per application," in *Proc. SAMOS*, 2011.

[8] B. Akesson *et al.*, "Predator: a predictable SDRAM memory controller," in *Proc. CODES+ISSS*, 2007.

[9] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. DAC*, 2010.

[10] S. Stuijk *et al.*, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *Proc. DSD*, 2010.

[11] G. Bilsen *et al.*, "Cyclo-static dataflow," *IEEE Trans. Signal Process.*, vol. 44, no. 2, 1996.