MOUSSE: scaling MOdelling and verification to complex heterogeneoUS embedded Systems Evolution

Markus Becker*, Gilles B. G. Defo*, Franco Fummi†‡, Wolfgang Mueller*, Graziano Pravadelli†‡ and Sara Vinco†
*Paderborn University, Germany, Email: {markus.becker, defo, wolfgang}@c-lab.de
†University of Verona, Italy, Email: {name.surname}@univr.it
‡EDALab s.r.l., Italy, Email: {name.surname}@edalab.it

***Abstract*—This work proposes an advanced methodology based on an open source virtual prototyping framework for verification of complex Heterogeneous Embedded Systems (HES). It supports early rapid modelling of complex HES through smooth refinements, an open interface based on IP-XACT extensions for secure composition of HES components, and automatic testbench generation over different abstraction levels.**

I. Introduction

The heterogeneity of modern Heterogeneous Embedded Systems (HES) implies different modelling languages, design strategies and optimization methods. Thus, the design of an HES cannot be based on a top-down coherent design methodology producing correct-by-construction implementations, but it must be based on the aggregation of pre-designed components through massive reuse of heterogeneous IPs [4]. Moreover, due to different abstraction levels, different aggregations can be assumed during the entire HES product life and new features must be also added, since some extensions, modifications and adaptations can be necessary to follow the market evolution. This further implies to complement the correct-by-construction methodology by a correct-by-evolution design approach based on new secure composition, seamless verification methods and tools which scale with the HES evolution. However, industry currently still suffers from the fact that there are no open integrated verification frameworks available for the prototyping of true HES. Several open source tools are highly advanced in efficiently managing models with high complexity, but they are limited to a single class of models of an HES [1], [2], [3].

In this context, we propose MOUSSE: an open source prototyping framework and a set of open source integrated tools to provide efficient secure composition and scalable verification methods for complex HES. To achieve such an ambitious goal, we have defined a methodology and, accordingly, we are developing a set of supporting tools focussing on the following sub-objectives:

1) Definition of techniques for HES modelling and meta-modelling.
2) Development of technologies and tools for accurate verification of HES at different abstraction levels.
3) Definition of techniques to abstract HES models according to a homogeneous computational model.
4) Development of technologies and tools for optimized accurate verification of the homogeneous model of the HES.
5) Correct-by-evolution abstraction/refinement of HES to support HES evolution towards more sophisticated versions during its lifetime.

After a section devoted to related works (Section II), this paper shows how our MOUSSE methodology, reflecting points 1-5, represents the basis to push forward the current limits of complex HES correct-by-evolution design and verification (Section III). Finally, experimental results (Section IV) and concluding remarks (Section V) summarize the main achievements and future activities of our work.

II. Related works

Component-based [4] and model-based [5] design flows are widely recognized as the main approaches for managing the complexity of modern embedded systems. They typically support reuse combined with a top-down design flow for heterogeneous components.

During the last years, the most path paving approaches were Ptolemy II and Metropolis. Ptolemy II [6] introduced a framework for modelling, simulation and design of concurrent, real-time, heterogeneous embedded systems. Metropolis [7], was designed to provide an infrastructure based on a meta-model with precise semantics, general enough to support existing computation models and to accommodate new ones. It has been shown that managing MoC heterogeneity is the main challenging task as it is also addressed by Ptolemy II and Metropolis and which we can also be found in Henzinger et al. [8], and in the Hybrid System Interchange Format (HSIF) [9], for instance.

Independently from the adopted MoCs or interchange formats, a different flow to design complex HES consists of modeling the system by using a system-level design language (SLDL), e.g., SystemC, SpecC [10], SystemVerilog [11]. In particular, SystemC with analog extension, SystemC-AMS [12], can cover heterogeneous domains of modern HES.In this context, some works have been proposed in order to either (i) extend the discrete event simulation semantics of SystemC to support multiple MoCs [13], (ii) introduce heterogeneous specification methodologies on top of the standard SystemC kernel [14], or (iii) combine C++, SystemC and SystemC-AMS to achieve interoperability [15]. Thus, to preserve component reuse, several other co-simulation frameworks have been described [16], [17], [18], where for the implementation of a virtual system prototype, an SLDL is applied in combination with an abstract canonical RTOS and/or an ISS, and/or a simulation engine for continuous behaviors. However, co-simulation assemblies heterogeneous components without providing a rigorous formal support, thus making integration and validation very hard tasks.

The work has been partially supported by the European Project SMAC (FP7-ICT-2011-7-288827).
978-3-9810801-8-6/DATE12/©2012 EDAA.

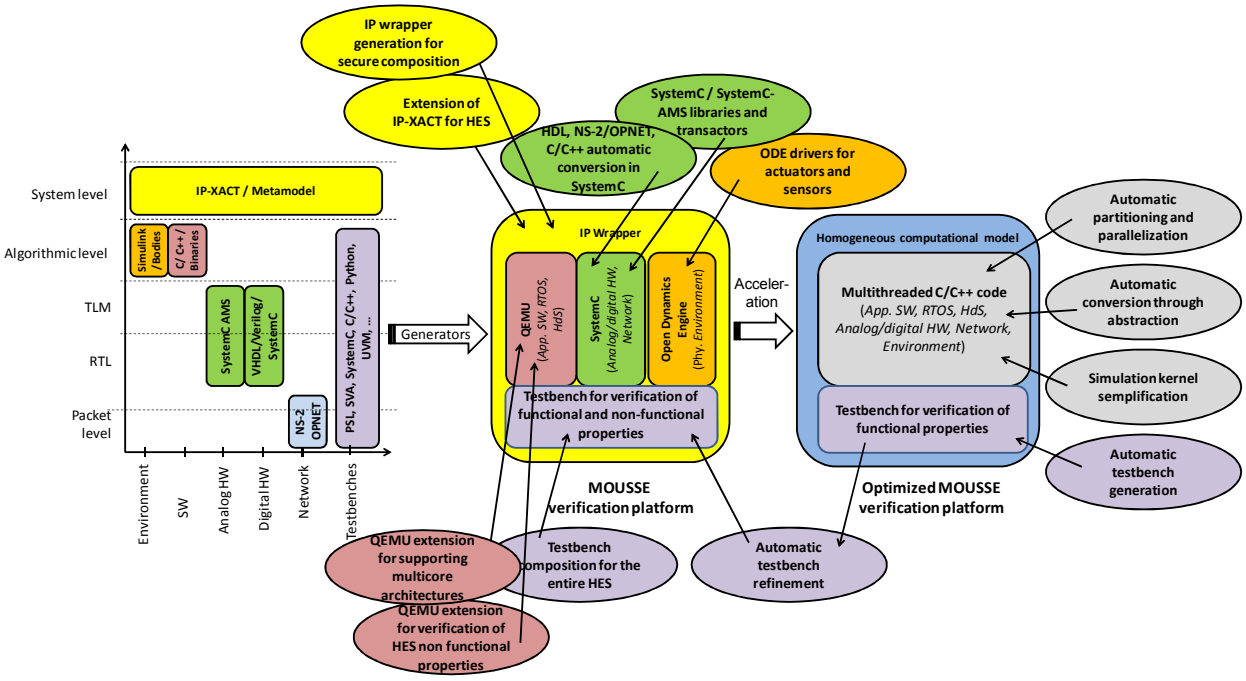

Fig. 1. The MOUSSE methodology.

III. Methodology

The key aspects of the proposed methodology are depicted in Figure 1. The starting point is represented by a set of heterogeneous modelling languages and open source tools which are generally used for modelling and aggregating the different components of an HES (Figure 1, left):

- IEEE IP-XACT is limited to composition of basic electronic system and requires significant extensions for HES application like for general software support and for power management. Based on UML metamodelling techniques, the MOUSSE project exploits an HES metamodel based on IP-XACT for secure composition.
- C/C++ is the obvious choice for modelling embedded SW due to its wide acceptance. Binaries for the target processor core can be managed as well as the framework integrates QEMU for efficient CPU and device emulation.
- SystemC and SystemC/AMS as an open-source IEEE standard for modelling digital and analog hardware components. Therefore, other high level SystemC libraries like TLM 2.0 and aRTOS [19] for RTOS abstraction can be easily included. Other IEEE standard languages (Verilog, VHDL, SystemVerilog) are included by conversion via the HIFSuite framework [20].
- Opnet and NS-2 are, respectively, market standard and open source framework for modelling the communication infrastructure; such models are in MOUSSE automatically converted into SystemC models based on the open-source SCNSL library extending the basic SystemC functionalities [21].
- Rigid physical bodies and their behaviour are modelled by using the open source Open Dynamics Engine, for the rapid prototyping of virtual physical environments, while it allows the MOUSSE project to produce a complete open-source framework for HES modelling and verification.
- Testbenches can be provided at different abstraction levels in different languages, such as PSL, SVA, SystemC, C/C++ and so on. By using the HIFSuite technology they are all concerted and represented in SystemC.

Starting from such an heterogeneous set of HES models, the first objective of MOUSSE is to provide translators and generators to create a set of semi-homogeneous executable platforms based on C/C++/SystemC models and some open-source libraries. These generator represent the *MOUSSE modelling framework*. The outcome of such generators represents the *MOUSSE verification platform* (Figure 1, centre), which is built as the integration of only three open-source tools: QEMU, SystemC and Open Dynamics Engine, as follows:

- The integration and synchronization of the three tools is based on the concepts of the TLM2.0 standard covering untimed, loosely timed, and approximately timed composition of components.
- As QEMU is developed for the fast execution of binaries on different target instruction architectures, significant extension for QEMU are identified for the verification of HES specific non-functional properties with focus on time and power analysis.
- For the integration of the Open Dynamics Engine (ODE) and its synchronization with TLM, dedicated ODE drivers for actuators and sensors for the analysis of HES functional and non-functional properties are generated.
- Though the current approach focuses on the integration and efficient synchronization of a heterogeneous set of three simulators, as an open framework based on novel metamodelling and code generation technologies, the platform is not limited to these tools and it is open for the integration and synchronization of other verification tools.

The second goal of MOUSSE is to increase the efficiency for verifying complex functional properties in the context of complex testbenches, e.g., for the exploration of corner cases, where some orders of magnitude speed-up are necessary

for these verification sessions. Such a simulation speed-up is obtained by the *optimized MOUSSE verification platform* (Figure 1, right), which is based on:

- Automatic generation of a homogeneous HES description based on a single simulation language for higher abstraction, and thus reducing the need for co-simulating and synchronization among different simulation engines.
- Automatic abstraction for the identification of HES abstraction levels dedicated to virtual prototyping. Hereby, the main goal is preserving the equivalence, but removing details which slow-down the simulation. This is done by abstracting data types, communication/synchronization and operating systems.
- Replacing simulation kernels with lighter kernel versions and using host OS primitives. This allows, for instance, to replace QEMU and Open Dynamics Engine by SystemC routines and to reduce the SystemC kernel to a set of subprograms calls [22].
- Parallelizing the code thus exploiting the powerful multi-threaded and/or multi-core execution architectures of the simulation host. This implies the transformation of SystemC code into raw parallel C/C++ code thanks to the exploitation of the homogeneous computational model.

The optimised MOUSSE verification platform is generated based on the univerCM homogeneous computational model [23]. In this way, MOUSSE is able to exploit powerful multi-threaded and/or multi-core architectures. Adequate scaling verification supports HES evolution.

Moreover, MOUSSE develops effective testbenches for the HES verification by:

- integrating functional testing, protocol conformance testing, software testing and analog testing techniques on top of a common mutation testing framework able to uniformly error simulating the HES [24], [25];
- exploiting the univerCM homogeneous computational model and the abstracted description of the HES to allow the exploration of the complete HES description;
- producing testbenches on this abstracted HES description and making them concretely executable on the simple verification platform;
- moving testbenches from the initial description to the abstracted ones and vice-versa as a further characteristic of the MOUSSE modelling and verification framework.

The integration of the modelling framework with the verification platform and the optimized verification platform constitutes the *MOUSSE virtual prototyping framework*.

IV. Experimental results

The MOUSSE project is still on-going, thus the different flows presented in Section Section III have not been completely integrated yet. In order to prove the effectiveness of the overall framework, experimental results are split into two cases: (i) automatic generation of homogeneous descriptions and automatic abstraction, to remove co-simulation and low level unnecessary details; (ii) RTOS abstraction. Future work will integrate the existing flows, in order to allow the management of more complex, heterogeneous descriptions.

A. HW module abstraction

The goal of this section is to show how the MOUSSE methodology can be exploited in order to reduce a complex

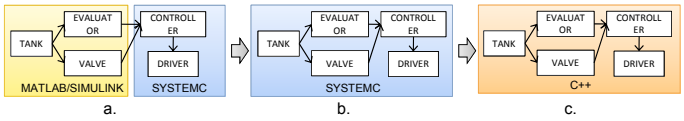

Fig. 2. The water tank system (a) and the system after the application of the MOUSSE methodologies (b-c)

Approach	S. time (s)	Error (%)
SystemC/Simulink cosim	7200.00	13.97
Homogeneous SystemC	202.45	2.08
Optimized SystemC	199.97	2.08
C++ abstracted code	6.11	2.08

TABLE I
Performance of the different simulation alternatives.

heterogeneous system to an homogeneous representation. The test case analyzed is a water tank system, combining analog (tank, valve and evaluator) and digital (controller) components and an HdS (device driver), as shown in Figure 2.a. This high level of heterogeneity would force the designers to use a co-simulation framework such as the one depicted in Figure 2.a, by integrating SystemC (digital controller and HdS) and Matlab/Simulink (analog components). Such a framework results in being really slow, as it is heavily affected by the extremely frequent synchronization steps caused by the fine-grain Simulink computation. Furthermore, there is no proof that the interaction between the different components is correct and the error with respect to the correct system behavior results non trivial.

By adopting MOUSSE, first all the components are automatically represented by using the univerCM computational model and then converted into a homogeneous SystemC description (Figure 2.b). Then, data type abstraction is applied to the generated code, in order to improve the performance by substituting the SystemC data type library with a more optimized implementation. Table I (line 2, 3) shows that simulation time (*S. time (s)*) is drastically reduced and that the simulation in more accurate (*Error (%)*), thanks to the homogeneous simulation environment. Finally, simulation time is decreased even more by abstracting the whole SystemC system into C++ code (Figure 2.c). This step outperforms all the previous optimizations, as the abstraction process allows to get rid of low level details and of the heavy SystemC kernel (line 4 of Table I). The accuracy in estimating the system behavior is still preserved.

B. RTOS abstraction

This section focusses on showing the feasibility and efficiency of MOUSSE software abstraction techniques. As a case study the original target firmware of a PowerPC405 example application has been abstracted at different levels of detail according to the refinement/abstraction flow depicted by Figure 3. The example application is composed of two computation intensive software tasks running at 100% utilization of the processor. Task 1 iteratively computes prime numbers. Task 2 recursively computes faculties of n. The task set is scheduled by the ORCOS[26] real-time operating system kernel using a fixed priority scheduler. The task execution is synchronized through kernel signals.

According to Figure 3 on a) the native SystemC level (i.e., the most abstract level) the complete firmware stack is modeled fully native in a homogeneous SystemC environment. The ORCOS kernel is abstracted by our native SystemC

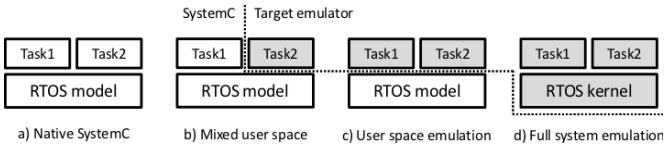

Fig. 3. Software refinement abstraction levels of the case study.

Lev.	Experiment description	S. time
a)	All tasks@SystemC w/ aRTOS	5.6s
b)	Task1@SystemC/Task2@QEMU user mode w. aRTOS	7.6s
c)	All tasks@QEMU user mode w/ aRTOS	9.2s
d)	ORCOS@QEMU full system mode	51.6s
d)*	ORCOS@QEMU full system mode w/ SystemC co-simulation in single-step synchronization	1472.2s

TABLE II
EXPERIMENTS ACCORDING TO THE ABSTRACTION LEVELS.

RTOS library aRTOS[27]. Communication is abstracted on the transaction-level and time is modeled on top of aRTOS by means of a software task synchronization statements at the granularity of source code branches. On b) the mixed user space level Task 2 was moved to QEMU user mode emulator whereas Task 1 remains in SystemC. For this, single application tasks can be co-simulated using our QEMU user mode wrapper providing an interface to the aRTOS core. On c) the user space emulation the complete application layer was moved to QEMU user mode. The RTOS/HdS layer still remains in SystemC. Finally, on d) the full system emulation the complete firmware stack was moved to QEMU full system emulator. Here, the original firmware can be simulated including register-accurate I/O device access from the RTOS/HdS layer. In order to distinguish the overhead of full system emulation from co-simulation overhead we split the performance measurement of Level d) into two experiments. The first experiment measures the overhead of QEMU full system emulation as a stand-alone simulation. The second experiments measures total overhead of running a SystemC kernel and the QEMU full system emulator in a single step synchronization, i.e., one target instruction per step.

Table II shows the comparison of the simulation overhead of the different abstraction levels by means of simulation time. We adopted the QEMU user mode and system mode emulators from the QEMU 0.12.1 release. Each application task was activated 1.000.000 times. All experiments produced the same task output thus proving a functional equivalence of the different abstraction levels. The results also showed that a significant simulation overhead can be saved by moving the HW/SW model step by step from a heterogeneous co-simulation environment to a homogeneous SystemC environment. Moreover, we were able to show that the execution speed especially of the wrapped user mode emulator of QEMU is close to native execution (1.5-2x) which results in pretty low simulation overhead compared to traditional ISS approaches that usually come with a slowdown of several orders of magnitude [28].

V. CONCLUDING REMARKS

This paper presented a bottom-up methodology for virtual prototyping of complex heterogeneous embedded systems supported by a set of open source tools based on SystemC, QEMU and Open Dynamics Engine. The tools are intended to be integrated in MOUSSE verification platform which provides a first environment for simulation and verification of HES. Then, abstraction strategies are applied to obtain the optimized MOUSSE verification platform where the HES is represented by multithreaded C/C++ code to further accelerate simulation. The methodology has been completely defined, some strategies and supporting tools have already been implemented and integrated in the MOUSSE virtual prototyping framework, while others are still under development. Experimental results have been focussed on abstractions of RTOS and HW components towards C/C++.

REFERENCES

[1] R. Smith, "Open dynamics engine v0.030 user guide," 2002.
[2] F. Ballard, "Qemu, a fast and portable dynamic translator," in *Prof. of USENIX Technical Conference*, 2005, pp. 41–46.
[3] OpenSystemC Initiative, "SystemC 2. 2 Language Reference Manual," 2007.
[4] E. Lee and A. Sangiovanni-Vincentelli, "Component-based design for the future," in *Prof. of ACM/IEEE DATE*, 2011, pp. 1–5.
[5] The MathWorks, "What is Model-Based Design," 2011.
[6] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the ptolemy approach," *Proc. of the IEEE*, vol. 91, pp. 127–144, 2003.
[7] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *IEEE Computer*, vol. 36, pp. 45–52, 2003.
[8] T. Henzinger, "The theory of hybrid utomata," in *IEEE LICS*, 1996, pp. 278 – 292.
[9] The University of Pennsylvania MoBIES Group, "HSIF semantics (version 3, synchronous edition)," 2002.
[10] D.D. Gajski and Z. Jianwen and R. Domer and A. Gerstlauer and Z. Shuqing, *SpecC: Specification Language and Methodology*. Springer, 2000.
[11] SystemVerilog Working Group, "SystemVerilog," URL: http://www.systemverilog.org.
[12] Open SystemC Initiative, "SystemC-AMS 1.0 standard," URL: http://www.systemc.org/downloads/standards/ams10.
[13] H. Patel, S. Shukla, and R. Bergamaschi, "Heterogeneous behavioral hierarchy extensions for SystemC," *IEEE TCAD*, vol. 26, pp. 765–780, 2007.
[14] F. Herrera and E. Villar, "A framework for embedded system specification under different models of computation in SystemC," in *Proc. of ACM/IEEE DAC*, 2006, pp. 911–914.
[15] M. Vasilevski, F. Pecheux, N. Beilleau, H. Aboushady, and K. Einwich, "Modeling refining heterogeneous systems with SystemC-AMS: application to WSN," in *Proc. of ACM/IEEE DATE*, 2008, pp. 134–139.
[16] F. Fummi, M. Loghi, M. Poncino, and G. Pravadelli, "A cosimulation methodology for HW/SW validation and performance estimation," *ACM TODAES*, vol. 14, pp. 23:1–23:32, 2009.
[17] S. Tudoret, S. Nadjm-Tehrani, A. Beneviste, and J.-E. Strömberg, "Co-simulation of hybrid systems: Signal-simulink," in *FTRTFT*, 2000, pp. 134–151.
[18] F. Bouchhima, M. Briere, G. Nicolescu, M. Abid, and E. Aboulhamid, "A SystemC/Simulink Co-Simulation Framework for Continuous/Discrete-Events Simulation," in *IEEE BMAS*, 2007, pp. 1 – 6.
[19] H. Zabel, W. Müller, and A. Gerstlauer, "Accurate rtos modelling and analysis with systemc," in *W. Ecker, W. Mueller, R. Doemer (eds.) "Hardware Dependent Software – Principles and Practice"*. Springer-Verlag, 2009.
[20] EDALab s.r.l., "HIFSuite," URL: http://www.hifsuite.com.
[21] F. Fummi, D. Quaglia, and F. Stefanni, "A SystemC-based framework for modeling and simulation of networked embedded systems," in *Prof. of IEEE FDL*, 2008, pp. 49–54.
[22] N. Bombieri, F. Fummi, and G. Pravadelli, "Abstraction of RTL IPs into embedded software," in *Proc. of ACM/IEEE DAC*, 2010, pp. 24–29.
[23] L. Di Guglielmo, F. Fummi, G. Pravadelli, F. Stefanni, and S. Vinco, "univerCM: the UNIversal VERsatile Computational Model for heterogeneous embedded system design," EDALab s.r.l., Tech. Rep., 2011, http://www.hifsuite.com/docs/univercm.pdf.
[24] G. D. Guglielmo, F. Fummi, G. Pravadelli, S. Soffia, and M. Roveri, "Semi-formal functional verification by EFSM traversing via NuSMV," in *Proc. of IEEE HLDVT*, 2010, pp. 58–65.
[25] G. Di Guglielmo, M. Fujita, L. Di Guglielmo, F. Fummi, G. Pravadelli, C. Marconcini, and A. Foltinek, "Model-driven design and validation of embedded software," in *Proc. of ACM AST*, 2011, pp. 98–104.
[26] Homepage of ORCOS: https://orcos.cs.uni-paderborn.de/orcos/. [Online]. Available: https://orcos.cs.uni-paderborn.de/orcos/
[27] H. Zabel and W. Müller, *Simulation with abstract RTOS Models in SystemC*, Dec 2007.
[28] H. Zabel, W. Mueller, and A. Gerstlauer, "Accurate RTOS Modeling and Analysis with SystemC," in *Hardware-dependent Software*, 2009, pp. 233–260.