An Heterogeneous Co-simulation Environment for Complex Embedded Telecommunication Systems

Fotios Gioulekas[1], Michael Birbas[1], Nikolaos Voros[2], George Kouklaras[2], Alex Birbas[1]

[1] Department of Electrical and Computer Engineering, University of Patras, Campus of Rion 26500, Greece. {fgiou, mbirbas, birbas}@ee.upatras.gr
[2]Intracom S.A. Telecommunications and Electronics Industry, 254 Panepistimiou Str. 26443 Patras, Greece. {voni, geko}@intracom.gr

Abstract

Bridging the different abstraction layers stemming from heterogeneous tools used in hardware software co-design for creating, maintaining and refining various system models form often a bottleneck in the overall design process. The missing test link between the specification level and the first refinement steps reveals the need for an environment that will enable the effective communication among the corresponding tools. This will also enable IP components to be reused early enough in the development cycle, even if their abstraction levels are different. This paper proposes a heterogeneous co-simulation approach to address this lack at this design stage. The overall result is an efficient environment that verifies and evaluates the conformance of the specification of a complex embedded system to its refinement.*

1 Introduction

A telecom system is classified to an important class of systems, called *reactive systems* [1], following the stimulus-response paradigm of behavior. Most co-design methodologies comprise a number of stages during the development and the final implementation of such systems often on a single chip (System-on-Chip). They always consider the requirement to reduce the design time without further increasing the total design cost. Figure 1 illustrates the conventional co-design steps. System's behaviour and special properties are specified employing formal languages (e.g. SDL [2]) or a variety of computational models [3, 4] at the specification phase. As part of the development process, it is needed to construct a virtual prototype before the final synthesis step.

The special characteristics of telecom systems, call for a co-design technique that enables the heterogeneous design and supports the co-simulation of the system's components at multiple abstraction levels. However, in most cases, developers employ cost-effective techniques that reflect their background experience; most of these techniques are based on proprietary platforms and tools [5]. Thus, there is not a generic technique and the proposed ones most of the times are inadequate for a heterogeneous development.

This missing link among heterogeneous environments and tools is an obstacle that forces the software and hardware groups to follow separate development paths. Thus, the overall testing is postponed for later stages, performing error-prone code generations before the synthesis step. Evaluation usually takes place through a time-consuming low-level co-simulation between a C-like language for software running on an Instruction Set Simulator (ISS) and VHDL or Verilog for hardware. Seamless CVE [6] is a typical co-design tool used in industry for co-verification purposes at this design phase, before physical electronic hardware is available. Coupling environments [7, 8]

* This work has been supported by EU through COSIBA project (IST-20688).

based on sockets to perform low-level co-simulation have also been proposed. However, the derived number of the late discovered errors and the increase of the number of the long iterations back to the partitioning or the specification phase increase the overall design time (see Figure 1) too. Thus, the need for interaction between software and hardware as early in the design as possible (in order to identify possible design errors) is necessary.

Within this context, an homogeneous and concurrent hardware and software development has been proposed in the literature employing SystemC [9] or SpecC [10] to add more detail at the specification level. In these environments, the C++ executable specification is simulated extensively and gradually enriched with hardware constraints. The next steps demand code generation from SystemC. Effective and powerful tools like ConCentric [11] are used to enable simulations at every refinement step. They also translate the SystemC model to synthesizable code in order to be fed to the prototyping board. However, although SystemC language constitutes a useful modeling platform, it is not widely accepted because it has not been standardized yet.

Other co-design tools, like Ptolemy [3] and PeaCE [12], target in concise specification capturing incorporating the appropriate Model of Computation (MoC). In that case, low-level co-simulation is employed after the design partition into C (software) and VHDL (hardware) target descriptions. The problem arises when the designer needs to couple different simulators since extra (and usually not trivial) development is required to modify appropriately their framework [13].

Similar techniques focus on moving from the specification abstraction level to the implementation level, performing homogeneous co-simulation on an ISS [14] or an HDL cycle accurate simulator (code generation is implied). However, the specification description is unable to interact with the potential refinements at lower abstraction layers.

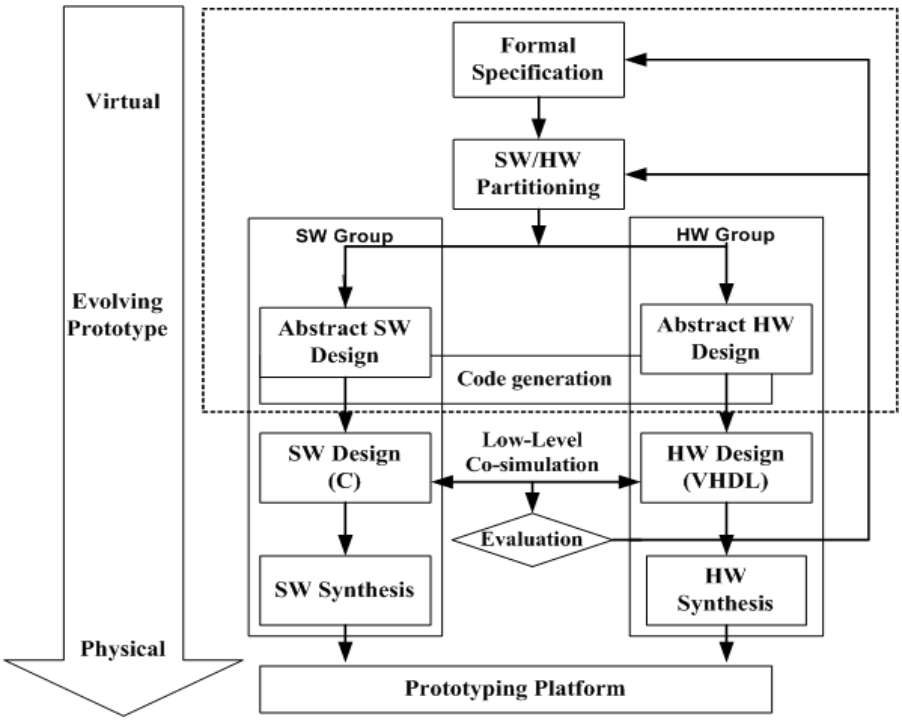

Figure 1. HW/SW Co-design flow

System-level design procedures are becoming a key part of development techniques. The decisions at the early co-design stages have the largest impact upon product cost, functionality and performance. The aforementioned environments offer functional system-level design but they perform code generation followed by low-level co-simulation for bridging different formalisms and tools. They often consider the ISS environment as the main platform for building their virtual prototype. This is an error-prone and time consuming procedure since the generated skeleton code

has to be enriched and possible logical errors lead to code redesign and debugging cycles. Thus, a mechanism to test the functionality of hardware and/or software (e.g. C/C++, HW-C based) modules with respect to specification descriptions (e.g. SDL based) **avoiding early code generation** is absent. The issue addressed by the proposed approach is depicted in Figure 1 bounded by the dashed line. Specifically, the approach presented here addresses the lack of a coupling environment for co-simulating heterogeneous parts of an embedded system, modeled in different abstraction levels, by providing the necessary linking mechanisms among the used Design Automation tools. It focuses on building a co-simulation environment coupling off-the-shelf microprocessor development tools and SDL CAD tools without changing the design flow. The proposed environment combines the semantics of SDL used in early specification stages and C running on an instruction set simulator at lower abstraction level.

2 Hardware/Software co-modelling using SDL and C

Embedded software tool vendors have given much attention in providing design platforms appropriate for co-modeling at an abstract level. These platforms support several possible refinement steps and imply mapping from higher to lower levels of abstraction [15]. During system-level design the necessary co-modeling of a telecom system comprised by hardware and software models, usually leads to a multi-lformalism approach. Since a network protocol is a reactive system, the mixed use of *FSM, Data Flow, Discrete Event and Imperative* computational models [3], is most of the times necessary to specify its control and data dominated behavior.

Our approach is based on heterogeneous modeling, adopting the benefits of SDL and C *standardized* languages to capture the semantics of the aforementioned models of computation. Both languages are well established in the telecom community and are supported by sophisticated tools. The communicating extended finite state machine model (CEFSM), implementing the SDL processes, is employed for the formal specification-description of the functional and dynamical body of a telecom system. (ETSI [16] provides SDL specification descriptions for a wide range of telecom protocols). Furthermore the additional usage of C language (not embedded in SDL), which is a textual imperative language that can be used to implement embedded software applications, express efficiently complex data processing. Therefore, by using these two formalisms the semantics and the properties of telecom applications are well defined. Figure 2 shows the proposed environment enhanced with heterogeneous co-simulation capability (between different abstraction levels) through the use of a mediator.

The objective is to allow the user to perform the design of a telecommunication system in an easy to handle environment. This will enable system functional verification and exploration at the architectural level with minimum overhead since the SDL-C model is already in an executable form, without having to address code generation issues. After the evaluation of the co-simulation results, this model could be targeted onto software or hardware implementation paths. Figure 2 illustrates the methodological steps towards the construction of a virtual prototype using the proposed approach.

- *Specification*. Following a top-down development, SDL is used to describe formally the telecom system at the specification level. Re-usable SDL components could be also adopted as part of system specification at this stage. Simulations, performed by using the available SDL simulator tool, evaluate the SDL description.

- *Refinement through functional decomposition.* At this step, the SDL specification is being refined. The telecom system's components that require heavily data processing tasks (e.g. CRC calculation, DSP algorithms) are implemented in C/HW-C running on ISS. The control-dominated behavior is maintained in SDL. Although SDL embeds the imperative model of computation, the usage of C language is more compact for representing this MOC and alleviates the SDL from dealing with complex tasks. If the co-design step of the low-level co-simulation has been performed and design errors have been identified, the environment could support a bottom-up approach: VHDL/C data dependent processes are described in C/HW-C while the control dependent ones are implemented in SDL.

• *Heterogeneous co-simulation.* Heterogeneous co-simulation enables disjoint semantics, which are present during the various design phases through the use of different formalisms, to interact. At this step, the derived SDL description and the C program running on the ISS are jointly tested, debugged and verified, thus performing heterogeneous co-simulation at different abstraction levels, while the error prone procedure of code generation is avoided (i.e. C code generation from the SDL description to be fed in the ISS does not have to take place). Both SDL and ISS CAD tools support simulation/execution engines that allow debugging and verification of the design. The communication between the two simulators is achieved through building well-defined interfaces (*link modules*) while a mediator administrates the co-simulation session [17].

However, the communication between those tools is not fixed. This gap had to be bridged in such a way that SDL and C models' semantics interact. This interaction had to conform to the communication mechanisms at the tools level by interchanging events. An SDL signal-event, sent to the environment, is mapped onto a C function used to describe the requested behavior. The data that the signal may convey are mapped onto C function arguments. The end (return) of a C function execution prompts the SDL environment to receive a signal with the appropriate parameters, if they exist. Therefore, a *discrete event model* of computation has been employed, which combines the MOCs of the selected languages. This adoption does not affect the properties of the modeled telecom system, since it describes part of its behavior. Moreover, the SDL signal and the return from a C function execution are mapped onto an event. At the simulators' communication level the master-slave scheme based on sockets has been adopted for exchanging information via a mediator. It uses a lock-step mechanism enhanced by special functions to raise incompatibilities between the client simulators and to conform to the discrete event MoC of the interface.

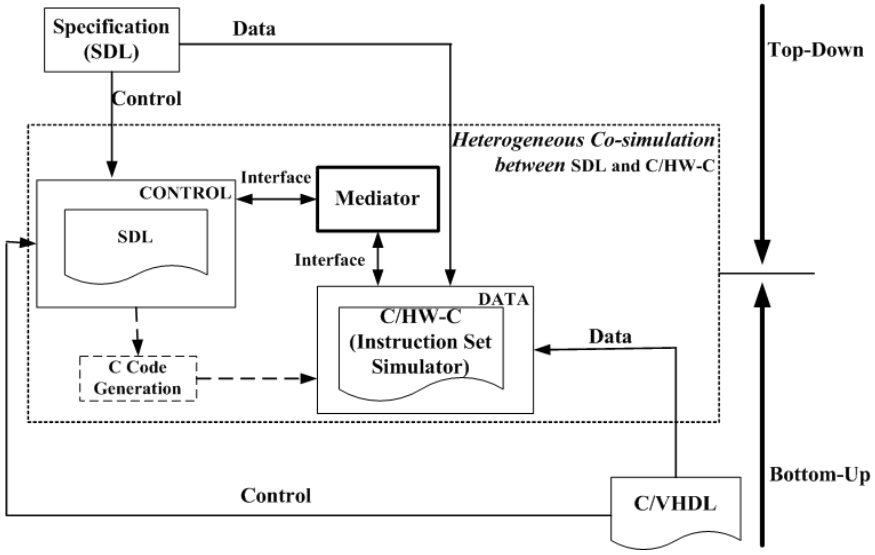

Figure 2. The heterogeneous co-simulation scheme

3 The co-simulation architecture

As aforementioned, the proposed approach aims to detect design errors close to the specification level, where the cost of the changes is lower, by performing heterogeneous co-simulation at different abstraction layers. Furthermore, the user has the chance to evaluate in parallel specification descriptions in SDL and SW/HW parts in C/HW-C running on an Instruction Set Simulator.

The proposed technique has been tested through the use of a specific ISS, the ARMulator[1] (the use of any other ISS could be also employed) and the Telelogic's TAU[2] SDT simulator. The co-

[1] ARM and the associated ISS (ARMulator) are Trademark of ARM Ltd. Corporation.

[2] Telelogic TAU is Trademark of Telelogic Corporation

simulation environment was built by performing the necessary extensions to the ARMulator and creating the appropriate synchronisation framework.

The SDL simulator (a discrete event simulator) communicates within TAU environment with other tools via a transparent API mechanism called Postmaster [18]. The ARM debugger is provided with a target description-the ARMulator- transparently attached to it. The ARMulator consists of four components: the processor core model, the memory, the co-processor and the operating system interface but does not provide a public interface. It can have two different levels of accuracy: *Instruction accurate* for accuracy at instruction boundaries only and *Cycle accurate* for guarantying the internal and external state of the model at every bus cycle. Since the ARM's debugger does not provide an API, the memory model, which is part of the ARMulator's environment, needs to be extended so as to provide the proper interface functionality.

The interface model of the co-simulation architecture is based on the sockets mechanism for achieving efficient communication between the SDL simulator API (Postmaster) and the custom extension in ARMulator's memory model. Since absolute accuracy is not needed during this design phase, the ARMulator is enhanced with discrete event characteristics. The distance between two consecutive events overlaps with the relevant boundaries as defined either by cycle or instruction accuracy levels.

The co-simulation mediator is attached to the client simulators via the aforementioned interfaces. Those *link modules* convey the information carried by the relevant simulator to the mediator and are based on sockets routines enabling RPC or IPC mechanisms. The co-simulation protocol that was developed is based on synchronous communication using blocking sockets mechanism in a lock-step manner. This synchronisation mechanism includes both postmaster's and socket's initialisation, transaction and termination routines. This algorithm is performed using the following procedures and is delineated in Figure 3.

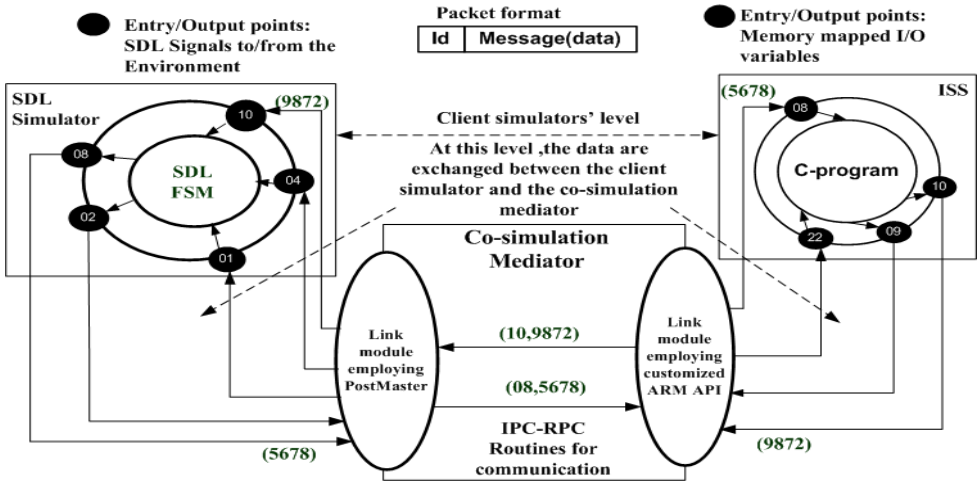

Figure 3. The co-simulation architecture

a) Scheduling. The user defines the client simulators' inputs. The SDL signal (event) name, which is the entry/output point to the SDL FSM, is assigned to a unique identity (*id*) for correct transmition via the sockets mechanism and reception by the foreign simulator. Moreover, the SDL signal name corresponds to a C function running on the ARMulator, which requests for data inputs (see Figure 3).

b) Synchronisation. The co-simulation environment formats the data and transmits them to the other client simulator through sockets using the socket transaction routine. The transaction between the client simulators is performed via the co-simulation mediator without need for intermediate storage. That achieves zero delay behavior, since the arriving events are not stored at the co-simulation environment. During this recursive process, if a simulator does not have data to send after an arriving message while the other client waits for data, the co-simulation mediator enforces it

to perform simulation, using a special message *"Ind"* in order to avoid deadlock. This environment doesn't support rollback mechanisms but guarantees, by using the blocking handshake socket mechanism, the non-violation of the *causality constraint* (reception of an event out of time-order) [19]. The client simulator is blocked in a certain state waiting for the arrival of an external message via the mediator's routines. The co-simulation environment is event accurate conforming to the SDL simulator's event accuracy and the ARMulator's instruction and cycle accuracy, as it is encapsulated in the extended discrete event behavior.

c) Transaction packet formation. The co-simulation mediator receives the SDL signal name and the parameter and identifies the unique number (*id*), defining the entry (output) graph points. The SDL simulator outputs the SDL signal parameters (data) with a special format depending on the data types it conveys and it is received using the postmaster's exported functions *SPRead* [18]. The co-simulation mediator treats this special format using a string-cut based algorithm that retrieves only the data types from the array in order to put them in the co-simulation packet. Each data type has a fixed size within the interchanged packet. During the formation of the packet, the algorithm appends to the received data the *id* and sends it over the sockets connection to the ARMulator side.

4 ARM ISS extension

To address the lack of an API for the ARMulator, we have extended appropriately the memory model [20] in order to embed a memory-mapped I/O peripheral functionality with socket connection capability. This port-mapped I/O communication scheme passes data between the bus and the ISS memory locations. Both the ISS and its memory model work with the same address and data bus. All mapped I/O locations in memory that are involved with co-simulation need to be identified in the C code running on ISS. The host system accesses these locations through a socket connection.

In the ARM-C program the I/O references to the external simulator are memory mapped to specific memory locations. When data arrive, the ARMulator's memory model traps a memory access and decodes the received packet. Firstly, it decodes the *id* field and identifies the function that the application should run. Moreover, it stores the data in special memory locations and when the specific C function is executed, it retrieves the data from those locations. In case that the application provides data to an external simulator, it stores them in other memory locations. The memory model traps another memory access and sends the data through sockets. Thus, the C application is simulated dynamically, enabling the functional verification of the developed protocol using different and multiple scenarios for its co-simulation. This functionality is embedded in a generic C *callback* function (for ARM ADS v1.1) or in a *memaccess* function (for ARM SDT v2.11a) [20]. Therefore, the extended memory model conforms to the handshaking synchronisation algorithm used during a co-simulation session. Figure 4 outlines this extension concept.

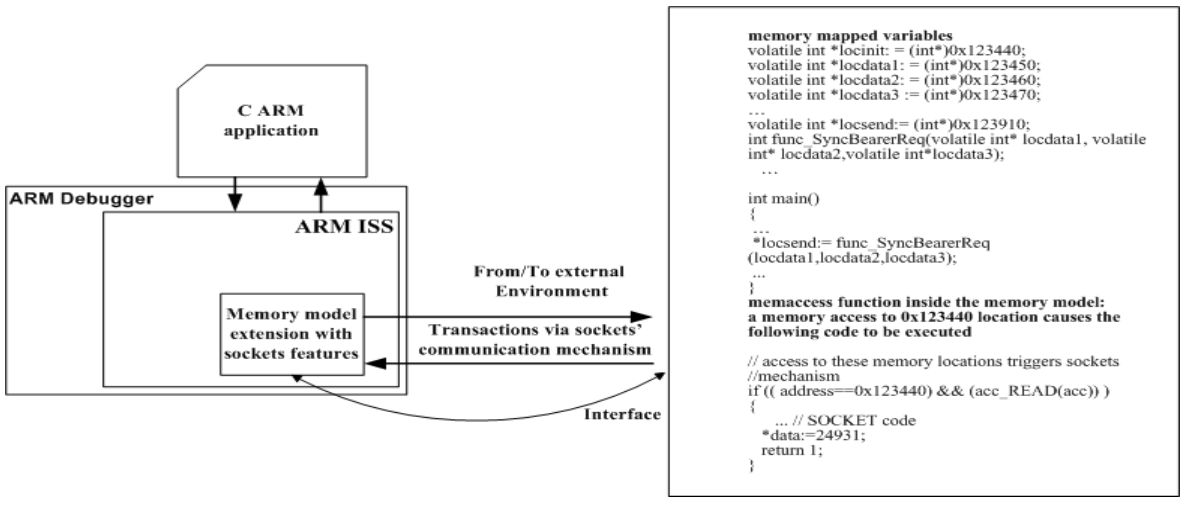

Figure 4. ARM ISS extension

5 Evaluation of the proposed technique through a case-study

The evaluation of the proposed environment has been performed according to a generic set of c*riteria* that are used to evaluate the effectiveness of most hardware/software co-design approaches. These criteria are *a) design time / cost-design iterations, b) user friendliness, c) re-usability support, d) used code overhead*.

The efficiency of the presented approach is exhibited through a case study that includes: (a) modeling and designing of the Medium Access Control (MAC) layer of the Digital Cordless Telecommunications (DECT) protocol stack [21], and (b) comparison to a previous development that was based on homogeneous high-level co-simulation [14, 22]. The evaluation and the comparison are performed based on the above criteria. Figure 5 compares the conventional design procedure followed in [14, 22] against the proposed one.

Specifically, we redesigned the DECT MAC system by following the proposed methodological steps. We have re-used SDL and C parts modelled in previous implementations [14, 22]. The Upper MAC, part of Lower MAC and the LLME (Lower Layer Management Entity) [21] have been implemented in SDL using the facilities of Telelogic TAU SDT environment. The Lower MAC procedures have been modeled in C running on an extended ARMulator memory model supporting sockets functionality for co-simulation. The ARM SDT 2.11a environment has been used for this development. The mediator co-simulated the SDL part with the C part running on the ARM ISS and thus C code generation and the Virtuoso operating system calls were avoided. The work in [14, 22] is given on the left hand-side of Figure 5 while the proposed approach is described on the right hand-side. It is clearly depicted that the presented methodology avoids the error prone C code generation from the SDL specification description.

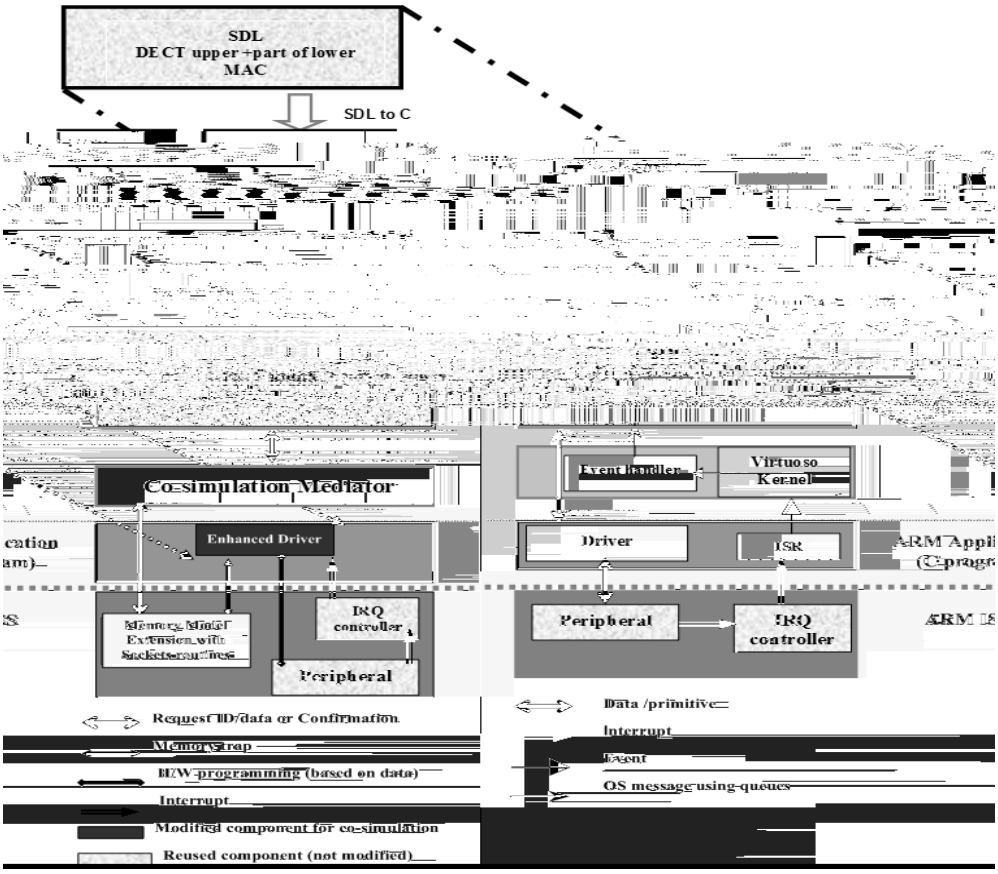

Figure 5. DECT case study

A. Design time/cost-design iterations. According to the development in [14, 22], early C code generation was produced manually from SDL descriptions (at the specification level). The generated code was introduced to the ARMulator, which incorporated HW-C models of the rest protocol's parts. Then, homogeneous co-simulation was performed at the same low-level of abstraction (ARMulator) between the software models described in C and the hardware models described in Hardware-C. The missing link between the SDT and the ARM Development Toolkit, forced the designer to follow separate development paths. Specifically, the SDL model was developed and simulated without taking into account the C and Hardware-C blocks, which were developed and tested separately. Therefore, the overall testing procedure was postponed for later stages including the error-prone code generation procedure.

The proposed approach, by employing heterogeneous co-simulation between different abstraction levels i.e. by jointly simulating the SDL specification description and the C application running on ARMulator, avoided the error code generation step. The duration of the established co-simulation session did not prolong the design cycle time. The measured IPC overhead was almost negligible (almost 0.013ms, where the co-simulation was performed on a Pentium 733MHz [17]). Table 1 depicts those results. The homogeneous co-simulation delay that was reported in [14] was 30 seconds. The time overhead with the proposed methodology depends on the involved mediator routines and the number of the exchanged messages. The different test cases that were performed in this case study determined the average total delay.

The number of late discovered errors was decreased, as well as the number of design iterations compared to the work in [14, 22]. Specifically, regarding the number of design iterations we had a reduction by 23%, while the number of late-discovered errors in the development cycle was reduced by 11% [17]. Those reductions referred to the avoidance of the errors introduced by the C code generation and its further enhancement using Virtuoso's services. The methodology proposed in [14, 22] had added an intermediate step in the co-simulation process. This intermediate step of C code generation from SDL had formed a significant overhead in this development cycle, since the generated C code skeletons were enriched manually. The derived design errors during this step had lead the designer to correct them in the SDL description. Thus, C code was re-generated and so on. The rest of the late discovered errors had to do with logical errors associated with the protocol's behaviour and with run time errors caused by the non-appropriate configuration of the used tools. Table 2 shows these results.

HETEROGENEOUS CO-SIMULATION	TIME
Average single message delay	0.013ms
Average total delay	40 sec

Table 1. Time overheads during co-simulation

B. User friendliness. An important step in our methodology is that it is able to provide various test cases with co-simulation scenarios that verify the implementation. In the context of the DECT development the relative test cases corresponded to Synchronisation, normal Receive/Transmit and Handover procedures. The objective was to ensure the system's conformance to those scenarios, to test the functional communication between the domains of different abstract characteristics (SDL and C) and to validate the protocol's correct operation. The Message Sequence Chart (MSC) [2], generated by the Telelogic TAU during the co-simulation session, was responsible for the observation of the selected test case scenarios. The user is also facilitated by having the ability to monitor the co-simulation session in parallel with the model's behaviour, while observing the MSC

and the ARM debugger. Figure 6 illustrates the MSC diagram corresponding to a part of a test case scenario.

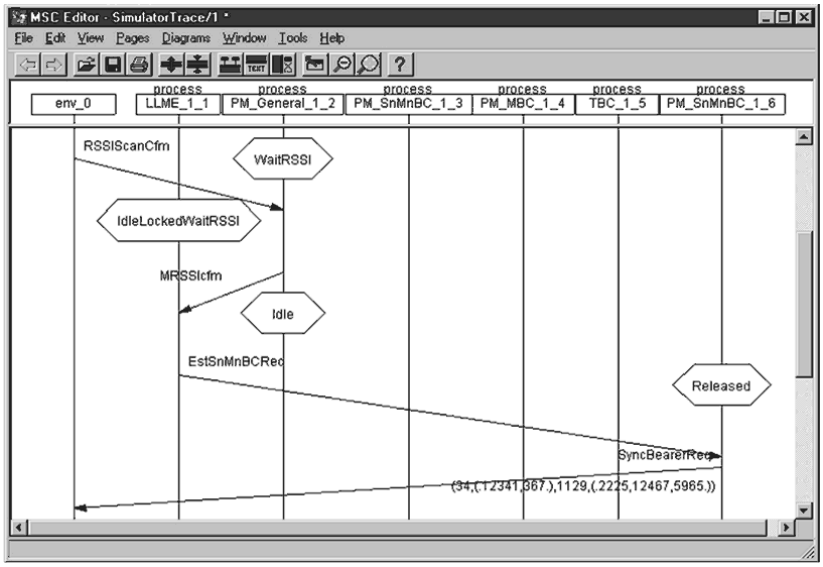

Figure 6. MSC diagram part of a test case scenario

C. Re-usability support. The SDL and the C model parts have been fully re-used (as described above). By following the conventional approach the concept of re-usability could not be supported, since the derived descriptions could not be re-used in a new SDL-based system design. As proven by applying the proposed approach to the specific case study, the ability to combine system-level IP reuse offered by our technique, facilitated distinct choices in the refinement procedure and enhanced the speed of the design cycle. If an error occurred in the model part described in C, this did not affect the SDL part. Thus, we did not need to change the SDL description functionality to perform co-simulation.

D. Used code overhead and expressiveness. The SDL-C model incorporates the appropriate MoCs that describe functionally a telecom system. The discrete-event mediator does not add extra behaviour to the system model and enables the observation of its dynamic characteristic. SDL and C support control and data-related behaviour under functional decomposition. The avoidance of the C skeleton code generation releases the designer from the need to perform manual code enrichments. The C code overhead, which is added to the C application running on the ARMulator and to the extended memory model, depends on the number of the memory mapped I/O variables. These variables are related to the number of the exchanged messages. This does not add to the design complexity.

CO-SIMULATION	CONVENTIONAL APPROACH [14, 22]	PROPOSED APPROACH
Design Iterations	89	68
Late discovered errors	52	46
Re-usability	Partially supported on the C model part	Full support (SDL, C)
Code overhead	Full dependency on C code generations and potential enrichments	Dependency on the number of the memory mapped variables

Table 2. Quantitative comparison between the proposed and the conventional design methodology

6 Conclusion

The approach presented here enables heterogeneous system level design using SDL and C formalisms, offering advanced reusability and heterogeneous co-simulation capability by bridging different levels of abstraction. Furthermore, it leads to a less error prone development cycle by enabling more accurate specification capturing and refinement derivation. Our intention was not to achieve full simulation speed compared to conventional low-level co-simulation approach but to provide a design environment improved in terms of clarity and understandability during the specification refinement steps. Obviously, the use of an alternative instruction set simulator instead of the ARMulator could be employed due to the generic nature of our technique. The concept of reusability, combining the semantics of hardware and software parts of a system early enough in the design workflow, is also supported. Moreover, the proposed approach provides an open architecture and it can be readily extended in order to incorporate SystemC or any other simulator in any abstraction layer (i.e. a VHDL simulator).

References

[1] A. Sarkar, R. Waxman, J. P. Cohoon, *"Specification-Modeling Methodologies for Reactive-System Design"*, High-Level System Modeling Specification Languages, Current Issues in Electronic Modeling, Kluwer Academic Publishers, 1995.

[2] SDL Forum. http://www.sdl-forum.org

[3] J. Davis II, M.Goel, C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. reekie, N. Smyth, J. Tsay, and Y. Xiong, *"Overview of the Ptolemy Project"*, Dept. EECS, Univ. California, Berkeley, CA, ERL Tech. Rep. UCB/ERL no. M99/37, July 1999.

[4] N. S. Voros, L. Sanchez, A. Alonso, A. Birbas, M. Birbas, A. Jerraya, *"Hardware/Software Co-design of Complex Embedded Systems - An approach using efficient process models, multiple formalism specification and validation via co-simulation"*, Design Automation for Embedded Systems, Vol. 8 Issue 1, Kluwer Academic Publishers, March 2003.

[5] Cavalloro P., Gendarme C., Kronlöf K., Mermet J., Van Sas J., Tiensyrjä K., Voros N. *"System Level Design Model with Reuse of System IP"*, Kluwer Academic Publishers, 2003.

[6] Seamless CVE. http://www.mentor.com/seamless/

[7] C. Valderrama, F. Nacabal, P. Paulin, A. A. Jerraya, *"Automatic Generation of Interfaces for Distributed C-VHDL Cosimulation of Embedded Systems: An Industrial Experience",* 7th IEEE International Workshop on Rapid System Prototyping, Thessaloniki, Greece, 19-21 June 1996.

[8] Wonyong Sung and Soonhoi Ha, *"Efficient and Flexible Cosimulation Environment for DSP Application",* IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences Vol. 0 pp 0-0 1998.

[9] SystemC Community. http://www.systemc.org

[10] SpecC. http://www.cecs.uci.edu/~specc

[11] ConCentric.http://www.synopsys.com/products/cocentric_systemC/cocentric_systemC.html

[12] PeaCE. http://peace.snu.ac.kr/research/peace/

[13] Y. Kim, K. Kim, Y. Shin, T. Ahn and K. Choi, *"An Integrated Cosimulation Environment for Heterogeneous Systems Prototyping"*, Design Automation for Embedded Systems 3, pp. 163-86, Kluwer Academic Publishers, 1998.

[14] S. K. Tsasakou, N. S. Voros, A. N. Birbas, M. V. Koziotis, D.G. Papadopoulos, *"High-level co-simulation based on the extension of processor simulators"*, Journal of Systems Architecture (47) 1 (2001), pp. 1-13, January 2001.

[15] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, A. Sangiovanni-Vicentelli, *"System Level Design: Orthogonalization of Concerns and Platform-Based Design",* IEEE Trans. CAD of Integrated Circuits and Systems, vol. 19, no. 12, DEC. 2000, pp. 1523-1543.

[16] http://www.etsi.org/getastandard/home.htm

[17] Fotios Gioulekas, Alex Birbas, "*Project results, experiences gained and lessons learnt*", COSIBA Deliverable IST-1999-20688/WP1/FEB03/D9, 05 February 03.

[18] Telelogic TAU™ Version 4.0 User's Manual, Chapter 11 *"The Postmaster"*.

[19] R. Fujimoto, "*Parallel Discrete Event Simulation Systems*", Communications of ACM, vol. 33, no.10, pages 30-53, October 1990.

[20] Application Note 32: The ARMulator (ARM DAI 0032C).

[21] DECT: The Standard Explained. February 1997.

[22] C. Drosos, M. Zayadine, D. Metafas, *"Report on the demonstration results of the dual-mode application using the Mobile/OS drivers' implementation",* D3.3R1 ARMOR ESPRIT Project (EP29251).