Cosimulation of Real-Time Control Systems

Juha-Pekka Soininen, Tuomo Huttunen, Kari Tiensyrjä and Hannu Heusala
VTT Electronics, Oulu, Finland

Abstract

The behaviour of a real-time system can be validated at the system level by means of a real-time operating system model in a VHDL simulation environment. The model consists of the real-time operating system, hardware described in behavioural VHDL and task descriptions written in C. The real-time operating system behaviour, HW/SW partitioning and HW/SW interfacing can be analysed by studying the simulation results.

1. Introduction

Technologies, methods and tools related to both the hardware and software of real-time embedded systems have developed rapidly. ASIC and FPGA technologies have become competitive implementation alternatives [1], and great progress has also been made with multi-purpose, RISC and DSP processors [2]. The target architectures of the systems are less obvious than before, because of the wider variety of possible technologies and the more obscure boundaries between them. An advanced application may contain embedded processors, ASIC's, FPGA's, real-time embedded software, etc.

Specification, partitioning and verification has increased significantly in importance, and main system properties, e.g. costs, performance and functionality, are greatly affected when an operating system is defined and task-level HW/SW partitioning is performed. System-level validation requires models that can be modified rapidly and analysed with advanced validation methods and environments.

HW/SW codesign has been widely recognised as a solution to the new requirements of system design [3], the idea behind it being to validate design decisions against system properties. The result is a better target system rather than highly optimised subsystems that do not operate properly together.

An implementation-independent specification language is a basic requirement for codesign. Simulation is usually carried out by translating the specification into VHDL or C. StateCharts [4], Structured Analysis (SA) [5] and SpecCharts [6] are examples of graphical source languages.

Validation of the functionality and performance of HW/SW partitioning is crucial part of any codesign environment. Since no implementation platform normally exists during the system-level design phase, the partitioned specification has to be executed or simulated using a more or less artificial implementation platform.

In prototyping environments [7,8] embedded systems are implemented using prototype boards that include programmable circuits, standard components and computer interfaces.

Several cosimulation approaches exist for the validation of HW/SW partitioning. In VHDL-C cosimulation the hardware part is simulated in a VHDL simulator and the software part executed as a C program. VHDL-C cosimulation is used in COSMOS and Vulcan cosynthesis systems, for example [9,10]. UNIX sockets and HW/SW communication primitives are used together with Verilog simulation in [11], while a processor model is used as an execution platform for HW/SW interaction in [12]. The Ptolemy framework uses a simulation backplane to combine simulation environments [13].

The COSYMA system uses run-time analysis for validation [14], whereas the TOSCA system uses synthesis results for the HW part and virtual assembly results for the SW part in architecture and performance evaluation [15]. A coarse-grain HW/SW partitioning is performed at the software task level in [16] and validated in [17].

More software oriented approaches are the physical modelling of software and RT-OS [18] and software simulation [19]. Monitoring tools have also been used to measure the performance of HW/SW systems [20].

The validation method presented here combines the cosimulation and software simulation approaches. The processor model in VHDL simulation is replaced with a real-time OS model. The software is executed as C program controlled by the simulated OS model, which is created using VHDL and C.

2. Validation of HW/SW partitioning in real-time embedded systems

Complex real-time embedded systems consist of a hardware part and a software part that runs under a real-time operating system. The hardware part consists of microprocessors, which may be embedded in ASIC, and application-specific hardware. The system-level validation method has to support the modelling of all parts of the system, including the behaviour of the real-time software.

2.1 Definition of the problem

The validation method for the partitioned model has to support functional validation, performance evaluation and architecture evaluation. The main problem is the analysis of software behaviour. The hardware designer can analyse the functionality and timing of his design by simulating the models, but in order to avoid the integration problems, similar methods and tools are needed for software and system design as well. This requires that the actual behaviour of the software and real-time operating system must be visible to the system designer during the partitioning phase.

2.2 VHDL platform

The analysis of system behaviour requires that the system specification is executed in some execution platform. In case the specification is done with an implementation-independent specification language, the execution platform should be an ideal machine. A VHDL simulator is a good candidate for such machine [21], and was chosen in our method for the following reasons:

Parallel execution. The VHDL has parallel structures, i.e. processes and signals. The ideal parallel machine can be used as a target architecture.

Control of timing features. Two separate time concepts exists in the VHDL simulator environment, simulation time and execution time. This makes it possible to model timing independently of the actual execution times in the simulation environment.

Use of foreign languages. The foreign kernel interface in VHDL'93 provides a way of importing algorithms written in languages other than VHDL into the simulation.

Possibility to use graphical front-end tools. VHDL can easily be generated from system-level descriptions.

Debugging and analysis features. VHDL simulators offer excellent debugging and analysis features compared with almost any other system. The VHDL language itself has many constructs that help in the design of analysis functions, e.g. assertion statements and analysis processes.

2.3 Real-time operating system modelling

In HW/SW cosimulation the software part is executed either on a workstation or in the simulated microprocessor model. The first approach is fast, but important data on the timing behaviour of the software model are lost. The second approach is practicable only when interface operations are being studied in detail. Otherwise the simulation is too slow.

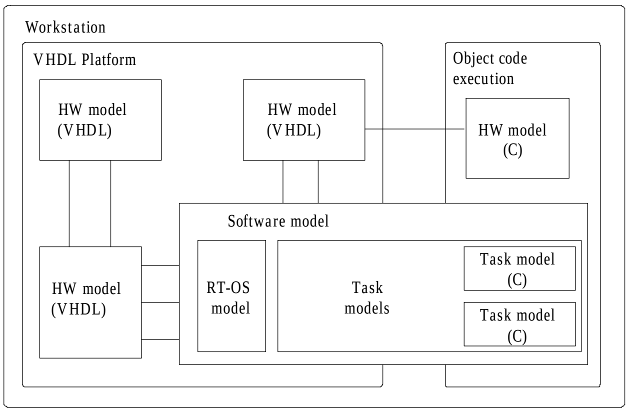

Figure 1. Architecture of the simulation model

In our approach the processor model is replaced with a reusable operating system model (Figure 1). The software is executed on a workstation, but the execution is controlled by the operating system modelled in VHDL. Communication between the operating system and the tasks occurs in the VHDL simulator, which can therefore monitor the interrupts, messages, switching of running tasks, task waiting times, etc. The simulation times are shorter than when using processor models as the execution platform. Some overhead occurs as compared with executing pure software on a workstation, because of communication between the VHDL simulator and the software model.

The approach can also be used for real-time operating system design and task allocation. The operating system model can be easily modified and optimised for the application being designed, and simulation of the system also allows rapid evaluation of the software architecture.

3. VHDL-based cosimulation of a partitioned system

Cosimulation of a partitioned system model entails simulating the hardware and software models together in order to study how the complete system operates, what the effects of the chosen partitioning are and whether the system fulfils its requirements.

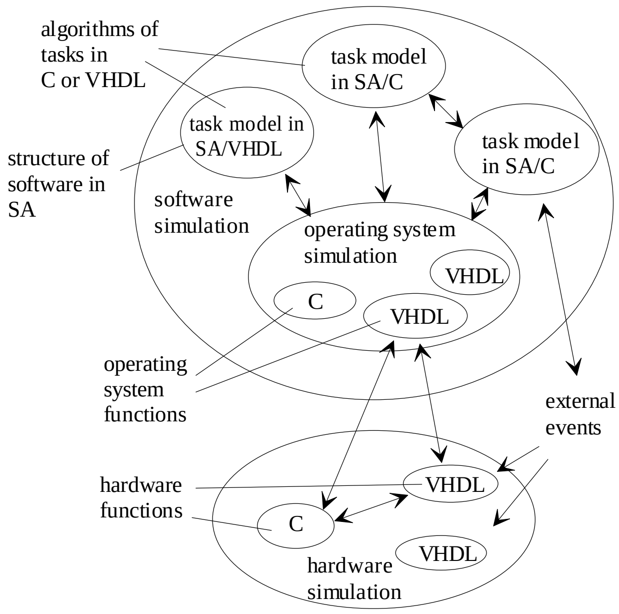

Figure 2. Structure of the simulation

The VHDL-based cosimulation approach is presented in Figure 2. The real-time system model is described by the SA/VHDL method [22], and the system structure using the SA hierarchy. The system functions are modelled by writing the algorithmic behaviour in VHDL or C. The parts described in C are linked to the VHDL simulation via the foreign kernel interface of the simulator. The elements communicate with each other by token passing.

The system model consists of hardware, software and operating system models. The operating system and software are tightly linked, since execution of the software tasks is completely controlled by the operating system and the software tasks and hardware communicate via the operating system. The use of C or VHDL is not limited by the location of the system function, and both SW and HW functions can be described in either VHDL or C.

3.1 Operating system model

The main operating system functions are modelled as data transformations, which are then described in C or VHDL, utilizing the parallel features of the later. Timers, for example, can be modelled as simple processes, and the actual implementation problems associated with software timers can be ignored.

The main emphasis is placed on task scheduling, since this has the greatest effect on behaviour. The scheduling function provides processing time for the tasks. The structure of the model (Figure 3) is based on the operating system, in which each task has a fixed priority. The key elements are the task list, containing the states of all tasks and timer 1, which acts as a system clock. Tasks can be inserted into the list when called up by an external interrupt or running task. The control scheduling part selects which task is executed next, the selection being based on the priorities. It also removes finished tasks from the task list.

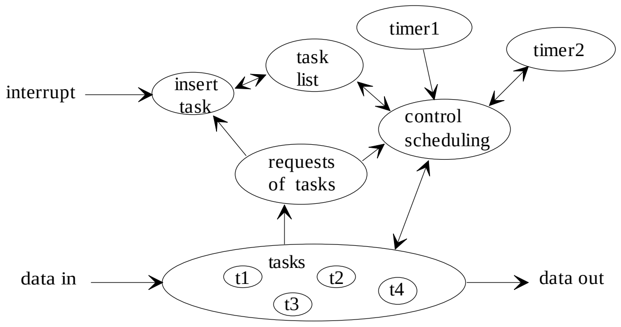

Figure 3. Structure of the scheduler

3.2 Software model

The software in the real-time control systems is organised in form of tasks that communicate via messages. In cosimulation, the model has to simulate this structure and behaviour. The SA/VHDL-based method involves modelling of the tasks as SA data transformations, which are specified in detail with C. In order to simulate the actual behaviour, however some changes have to be made to the task descriptions concerning state behaviour and execution control.

Each task has three active states, i.e. ready, waiting and running, the transitions between these states being controlled by the operating system. In addition to the states, the algorithmic behaviour also has to be modelled. The algorithms are described using C or VHDL, but the problem with algorithm description and simulation is that the simulation and execution times are different, so that the data values used in an algorithm may not be valid, since they may have been changed by some other, parallel activity. This can be avoided by dividing the algorithm into phases according to the data inputs and outputs (Figure 4). Execution of the algorithm during the simulation is suspended when the communication point is reached, until enough simulation time has elapsed, after which it proceeds with updated data values.

The outputs of the algorithm are sent only after the simulation time has elapsed.

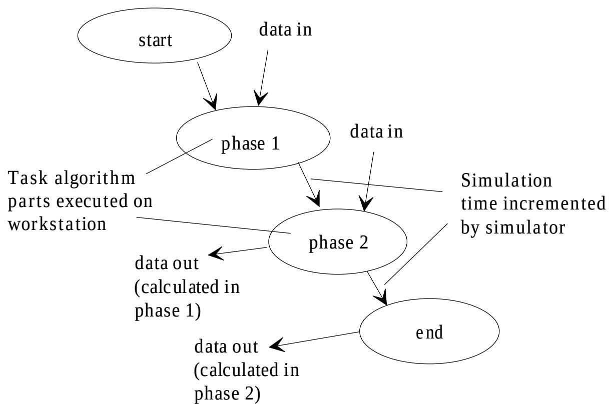
start
data in
phase 1
data in
Task algorithm parts executed on workstation
Simulation time incremented by simulator
phase 2
data out (calculated in phase 1)
end
data out (calculated in phase 2)

Figure 4. Structure of the algorithmic behaviour of a task

The task models written in C are linked to the VHDL simulator via its foreign kernel interface (FKI), in which the object code of the algorithm is linked to the simulator with a process description. The algorithm consists of two parts, the first, which is executed only once during the simulation, describing the input and output signals and memory allocations, while the second part is the executable algorithm, which is executed when there is an event or transaction in any of the input signals. The algorithm itself can be written with standard C. The VHDL/C interface is implemented using simulator kernel functions.

Communication between the tasks constitutes an important part of the software model. In real-time operating systems this communication takes place via control and status messages. The most common control messages deal with the starting and stopping of the tasks, which ask the operating system for services by means of status messages. The messages are modelled as data flows, which are then converted to VHDL signals when the model is translated into VHDL.

3.3 Modelling of hardware

The hardware model is an abstract functional specification written in SA/VHDL. The structural information contained in it presents the functional structure of the system and not its physical structure. The descriptions of the algorithmic behaviour of the functional units can also be very abstract, since the idea is to validate the behaviour and not to design the structural implementation.

3.4 Simulation

The original model was created using the SA/VHDL method extended with minispecifications written in C. The SA structure can be translated automatically into VHDL with the Velvet tool [22]. The C descriptions are translated into object code, which is automatically linked to the simulation model by the simulator. The VHDL simulator and the modelling practices adopted give full control and observability with respect to the actions of the operating system and software tasks, e.g. scheduling, the task list and message passing can be monitored from waveform displays.

4. Experiment: the Ethernet bridge

The approach was tested with the Ethernet bridge example [23], the function of which is to transmit data packages from one segment to another. In order to minimise the net load, the bridge has a capability for learning the network structure. When the bridge receives a package from a segment, it updates its database by linking the source address to the segment information. It then checks whether the target address is in the database and the packet is either filtered or sent to a specific segment or to all segments. Optimisation of operation requires that the bridge should have some buffering capacity.

4.1 Bridge architecture

The architecture of the implementation consists of three Ethernet cards connected to an 8051 micro-controller. It is not the intention here to optimise the features of the real system but to demonstrate the cosimulation method. The physical transmission and buffering of data packets is handled by the Ethernet cards, while the forwarding and filtering functions of the bridge are implemented in the 8051 as software tasks in RMX operating system.

4.2 Partitioned model

The partitioned model was created by the extended SA/VHDL method, in which the software part was divided in two: the operating system model and the task model (Figure 5). The complete model consists of 17 data transformations described in VHDL and 11 tasks and 6 subfunctions described in C. The simulated description consists of 7000 lines of VHDL code and 1000 lines of C code.

The modelling was done on a Sun Sparcstation 2, the translation from SA/VHDL/C into VHDL/C with

Velvet taking about one minute, the analysis with VHDL2000 about 10 minutes and simulation of the transfer of one packet to another segment about 5 minutes.

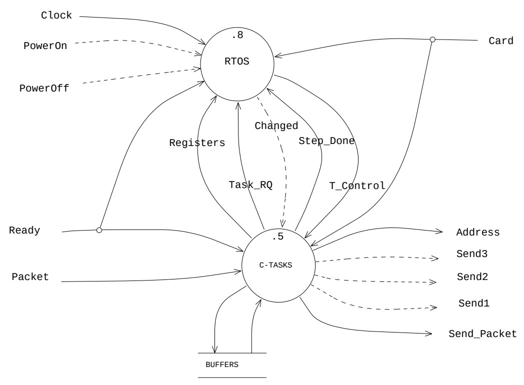

Figure 5. Software part of the SA/VHDL model

4.3 Results

The model was tested with simulations, a typical simulation result of which is presented in Figure 6. The figure shows a situation where a data packet is received (first marker) and then filtered (second marker). Finally it is transmitted and the buffer is released (third marker).

The experiment demonstrated that it is relatively easy to model the operation of the RT-OS commonly used in real-time embedded controllers. This RT-OS can be separated from the control software and reused in other designs.

The RT-OS modelling approach is compared with the use of functional processor models and standard VHDL-C cosimulation in Table 1. Functional processor models can be used for very accurate system-level validation when simulation time is not critical and when a target processor and software exist. VHDL-C cosimulation is a quick way to analyse HW/SW partitioned descriptions providing that an executable code exists and that the target RT-OS can be emulated in a workstation environment. The RT-OS modelling approach presented here is a compromise that provides accurate results with a reasonably short simulation time. The possibility for analysing the model without having completed the system description also extends the usability of cosimulation during system-level partitioning.

The modelling approach can also be used as a fast prototyping method. If the execution times of tasks in the real target environment can be estimated properly, the method even exceeds the features of a traditional prototyping environment, because simulation is independent of the execution environment. The analysis features of the VHDL simulator can effectively be used to study the behaviour of the software system. Possible uses are the design of operating systems, prioritisation of tasks and analysis of HW/SW partitioning.

Table 1. Comparison of cosimulation approaches

	Functional processor model	Software execution on workstation	Operating System model
Simulation time	very long	short	moderate
Model accuracy	RT/pin level	depends on workstation	functional level
Hardware model	µP based model	not needed for µP	not needed for µP
Software model	compiled code	executable code	behavioural model

The current approach nevertheless has some limitations. It is difficult to model data-dependent execution times for tasks, and the extent of external communication of tasks also affects the simulation time. At the same time the complexity of the operating system influences the efficiency of the simulation.

5. Conclusions

This paper describes a method for modelling and validating HW/SW partitioned real-time embedded control systems. The method uses a real-time operating system model to control the execution of real-time control software. The system model is described using the SA/VHDL specification language extended with C, and the system is validated using VHDL as a technology-independent implementation platform. All the debugging and analysis features of the VHDL environment familiar to the hardware designer are available for the analysis of the behaviour of the real-time control software. The method is suitable for the validation of system-level HW/SW partitioning and for the development of small real-time operating systems and control software.

6. Acknowledgements

This work was carried out within the ESPRIT basic research project EP8135 (COBRA, Codesign Basic Research Action). The work was supported by Technology Development Centre of Finland, TEKES.

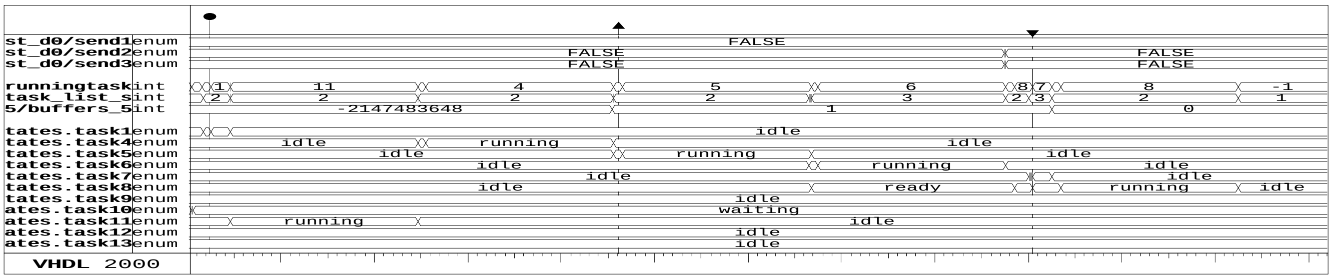

Figure 6. Simulation results

References

[1] Gupta, R K., Coelho, C. N. and De Micheli, G.: Program implementation schemes for hardware-software systems, *Computer*, vol. 27(1), 1994, pp. 48-55.

[2] Franke, D.W. and Purvis, M.K.: Hardware/Software Codesign: A perspective, *13th Int. Conf. on Software Engineering*, IEEE CS Press, Los Alamitos, California, 1991, pp. 344-352.

[3] De Micheli, G.: Computer-Aided Hardware-Software Codesign, *IEEE Micro*, Aug. 1994, pp. 10-16.

[4] Harel, D.: StateCharts: A Visual Formalism for Complex Systems, *Science of Computer Programming*, vol. 8, 1987, pp. 231-274.

[5] Ward, P.T. and Mellor, S.J.: Structured Development for Real-Time Systems, Yourdon Press, vol. 1-3, Englewood Cliffs, N. J, 1985.

[6] Narayan, S., Vahid, F. and Gajski, D.: Translating System Specification to VHDL, *Proc. of European Conf. on Design Automation Conf.*, IEEE CS Press 1991, pp. 390-394.

[7] Koch, G., Kebschull, U. and Rosenstiel, W.: A Prototyping Environment for Hardware/Software Codesign in Cobra, *Proc. of 3rd Int. Workshop on Hardware/Software Codesign*, IEEE CS Press, Sept. 1994, pp. 10-16.

[8] Buchenrieder, K. and Veith, C.: A Prototyping Environment for Control-Oriented HW/SW Systems Using State-Charts, Activity-Charts and FPGA’s, *Proc. of European Design Automation Conference*, IEEE CS Press, Grenoble,France, Sept. 1994, pp. 60-65.

[9] Ismail, T. B., Abib, M. and Jerraya, A.: COSMOS, A CoDesign Approach for Communicating Systems, *Proc. of the 3rd Int. Workshop on Hardware/Software Codesign*, IEEE CS Press, Grenoble, France, Sept. 1994, pp. 17-24.

[10] Gupta, R. and De Micheli, G.: Hardware/Software Cosynthesis for Digital Systems, *IEEE Design & Test of Computers*, Sept. 1993, pp. 29-41.

[11] Thomas, D.E., Adams, J.K. and Schmit, H.: A Model and Methodology for Hardware-Software Codesign, *IEEE Design & Test of Computers*, Sept. 1993, vol. 10, No. 3, pp. 6-15.

[12] Kumar, S., Aylor, J. H., Johnson B.W. and Wulf, Wm. A.: A Framework for Hardware/Software Codesign, *IEEE Computer*, Dec. 1993, pp. 39-45.

[13] Kalavade, A. and Lee, E.A.: A Hardware-Software Codesign Methodology for DSP Applications, *IEEE Design & Test of Computers*, Vol. 10, 1993.

[14] Ernst, R., Henkel, J. and Benner, T.: Hardware-Software Cosynthesis for Microcontrollers, *IEEE Design & Test of Computers*, Vol. 10, No. 4, Dec. 1993, pp. 64-75.

[15] Antoniazi, S., Balboni, A. and Fornaciari W. and Sciuto D.: A Methodology for Control-Dominated Systems Codesign, *Proc. of the 3rd Int. Workshop on Hardware/Software Codesign*, IEEE CS Press, Grenoble France, Sept. 1994, pp. 2-9.

[16] D'Amrosio, J. G. and Hu, X.: Configuration-Level Hardware/Software Partitioning for Real-Time Embedded Systems, *Proc. of the 3rd Int. Workshop on Hardware/Software Codesign*, IEEE CS Press, Grenoble France, Septemper , 1994, pp. 34-41.

[17] Sipola, M., Soininen, J-P. And Kivelä, J.: Systems Real-Time Analysis with VHDL Generated From Graphical SA-VHDL, *Proc. of 2nd European Conf. on VHDL Methods*, EUROASIC'91, Stockholm, Sweden, Sept. 1991.

[18] Alonso, A., Elmstrøm, R., Pezzè, M. And Pulli, P.: Outline of the Physical Model, IPTES-project EP5570 Report, 1993, 32 pages.

[19] Leskelä, J, Salmela, M., Heikkinen, M. and Hyvärinen, J.: Visualisation of Real-Time Software in Host-Based Simulation Environment, *Euromicro Workshop on Real-Time Systems*, Odense, Denmark, June 1995.

[20] Kalinsky, D. and Avnur, A.: Computer Aided Real-Time Design, *3rd Israel Conf. on Computer Systems and Software Engineering*, 1988,pp. 4-13.

[21] *IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987*, New York, USA, March, 1988.

[22] Kauppi, M.J. and Soininen, J-P.: Functional Specification and Verification of Digital System by Using VHDL Combined with Graphical Structured Analysis, *Proc. of 2nd European Conf. on VHDL Methods*, EUROASIC'91, Stockholm, Sweden, Sept. 1991.

[23] Backes, F.: Transparent bridges for interconnection of IEEE 802 lans, *IEEE Network*, Vol. 2, No. 1, 1988.