Towards the Creation of an ECU Model Exchange Market

Murakami, Kazuaki
Kyushu University

Yoshimatsu, Norifumi
ISIT

Rao, Pradeep
ISIT

Oho, Shigeru
Hitach Ltd.

他

出版情報：ICROS-SICE International Joint Conference. 2009, 2009-08
バージョン：
権利関係：

Towards the Creation of an ECU Model Exchange Market

Kazuaki Murakami[1,2], Norifumi Yoshimatsu[2], Pradeep Rao[2], Shigeru Oho[3] and Satoshi Shimada[4]

[1] Kyushu University, Fukuoka, Japan (Tel : +81-092-802-3792; E-mail: murakami@ait.kyushu-u.ac.jp)
[2]ISIT, Fukuoka, Japan (Tel: +81-092-852-3460; E-mail: {nyoshimatsu, pradeep.rao}@isit.or.jp)
[3]Hitach Ltd., Tokyo, Japan (Tel: +81-042-323-1111; E-mail: shigeru.oho.ku@hitachi.com)
[4]HONDA R&D, Tokyo, Japan (Tel: +81-028-677-3377; E-mail: Satoshi_Shimada@n.t.rd.honda.co.jp)

Abstract: Model based design can be used to address the challenges brought about by complex ECU designs. This paper introduces the use of models at various stages of ECU design and highlights their advantages. We particularly focus on the use of software models in ECU design and discuss their classification based on their use in the embedded system design flow. We discuss some of the issues involved in the use of the software models in ECU design and identify directions for research to facilitate greater use of software models in mainstream ECU design.

Keywords: simulation, model based design

1. INTRODUCTION

Embedded system designs involving electronic control units (ECU) continue to demand higher reliability and functionality while market demands continue to shrink the development time. The requirement to support advanced features make ECUs increasingly complex. For instance, many automotive ECUs implement functionality that increase safety, provide greater comfort and improve energy efficiency. This increase in complexity compounded by constantly changing specifications, results in longer development cycles and leads to greater costs and adversely impacts reliability.

Embedded system design specifications often require both the software and the hardware to be designed. In such cases, since the hardware platform is initially unavailable, software development has to wait until the last stage of the system design (when the hardware design and associated tools are implemented and available for software development). This may result in poor quality of the embedded system or software bugs being detected after system deployment. Even when a mature hardware platform is available for software development, current development techniques are expensive as they are based on the use of *test-boards*. Development flows based on the use of test-boards also prevent effective collaboration between software development teams that are geographically distributed.

Model-based designs (MBD) [10] are widely used in ECU development to address the above issues. In MBD, applications can be described under different levels of abstraction while preserving their specifications and are employed in the V-process for the design of ECUs. In the V-process, a different model is used for MBD at each level of the process while refining and testing the design and is shown graphically in Fig. 1. These models can be implemented in software, hardware or a combination of both. Software models representing the hardware are increasingly used for simulation as they facilitate software development before the actual hardware platform is available and thus help reduce the design time. Software models also enable safety analysis (fault tree analysis and failure modes and effects analysis), timing analysis (processor response and bus timing) and formal verification of system behavior [2]. These analyses enable the detection of critical design problems earlier in the ECU design cycle.

Despite these advantages in using software models, the existing models are insufficient for ECU design [10]. To encourage wider use and availability of software models in ECU design and analysis, we propose the creation of an ECU model exchange market. The purpose of the ECU model exchange market is to facilitate model development and its availability for ECU design – especially for the software based models of the ECU system. To achieve this, the models need to balance simulation speed and model accuracy while keeping development costs low. This paper reviews the existing approaches to modeling ECU based systems and discusses directions for further research.

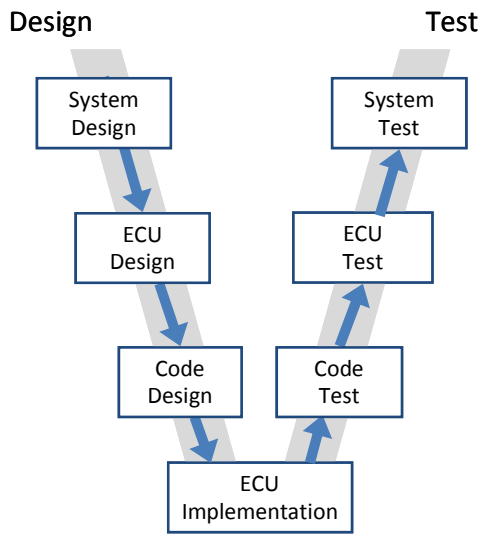

Fig. 1 V-process

We begin by discussing the embedded system design flow in the following section and highlight the use of models in various design stages. In section 3, we describe software models in particular and identify a few issues that need to be addressed with respect to

software models representing ECU platforms. In section 4, we propose approaches to address the issues raised and document related work in Section 5. We conclude in section 6 with our plans for future work.

2. EMBEDDED SYSTEM DESIGN FLOW

We first introduce the typical flow adopted in the design of embedded systems [1]. An embedded system design realizes its functionality through specific design phases – starting from specification definition and progressing through to its implementation. While refining the embedded system design, the following stages are typically employed.

- Specification definition
- Algorithm design
- Software design
- Object code design
- Implementation

We describe each of these refinement stages and also describe the appropriate models and the simulation methodologies employed in each of these stages.

2.1. Specification definition

The performance goal and functionalities to be realized in an embedded system is defined as the specification. During specification definition, the requirements and specification of a target system is described in a readable form.

2.2. Algorithm design

In this step the algorithms are designed to meet the specification defined in the previous stage. The algorithms can be designed without considering the specification of the devices (including processor architectures in a target system) or its timing behavior.

To facilitate rapid development, the algorithms are described using block diagrams or a model programming language. The model programming language supports the description of the algorithm at a higher abstraction level than the programming languages used to implement the target system. The environment that supports the model programming languages also support sensor, actuator and the plant/mechanical models to facilitate evaluation of the complete control system (e.g. MATLAB/Simulink etc.).

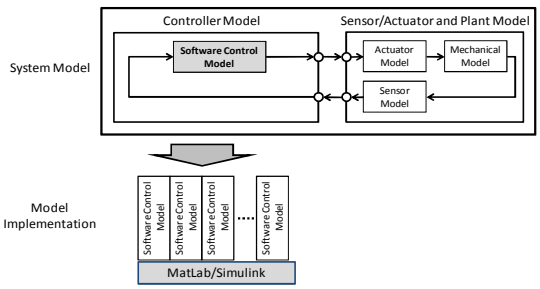

Fig. 2.1 Simulation model for MILS

The simulation using such a model description is called MILS (Model In the Loop Simulation). Fig. 2.1 shows a system model which consists of the c*ontroller model* and the *plant model*. The plant model typically consists of actuators that drive the plant and sensors that monitor the plant along with the mechanical model that represents the controlled system. In MILS, the software controller model is realized using a model programming language. The software controlled model is evaluated by compiling the model and executing it.

2.3. Software design

In the software design stage, the algorithms are implemented using a programming language. However in this refinement step, the program source code can be designed without considering either the specifications of the devices in the target system or the timing constraints, as in algorithm design.

The simulation used in software design step is called SILS (Software In the Loop Simulation). Fig. 2.2 shows a system model for the software design step. The controller model is represented by a program source code.

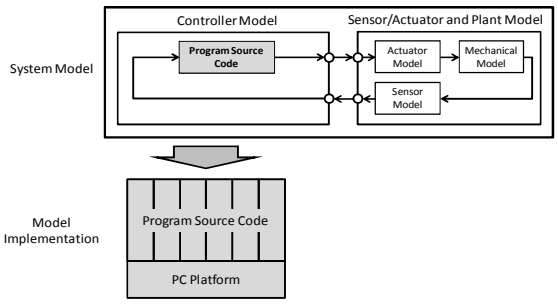

Fig. 2.2 Simulation model for SILS

2.4. Object code design

In this stage the program source code is compiled into object code while taking into account the precise specifications of the target system's hardware as well as its timing constraints. There are two approaches for simulations used in this stage.

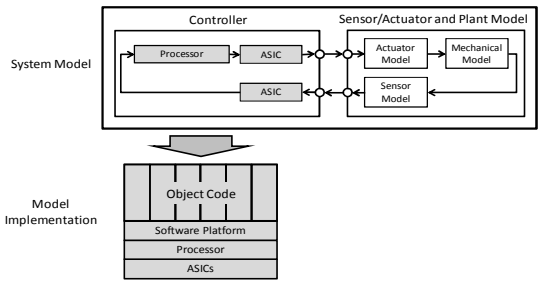

Fig. 2.3 Simulation model for PILS

- PILS (Processor In the Loop Simulation) uses the actual hardware of a target (involving processors as well as ASICs) as the 'Controller'. Fig. 2.3 shows a system model of PILS.
- SPILS (Software-based Processor In the Loop

Simulation) simulates the target platform. Here, both the application program and the hardware is described in software and the entire system operation is executed as a software simulation. The simulator uses software to model the behavior of the processor and ASICs in the target system as well. Fig. 2.4 shows a system model of SPILS.

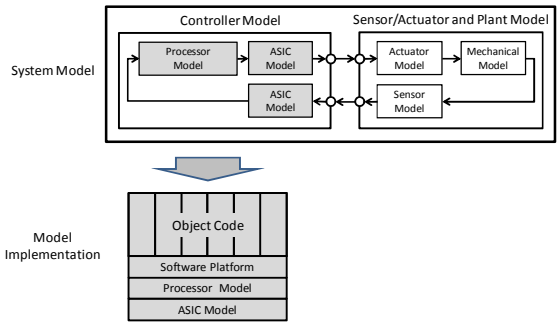

Fig. 2.4 Simulation model for SPILS

Both in the PILS and in the SPILS, the program object code is loaded into the program memory on the controller and executed. This stage requires precise specification of the hardware and also requires the representation of precise timing behavior.

PILS uses actual hardware and hence, exact functionality and timing behavior is guaranteed. In SPILS, the accuracy of the functionality and timing behavior depend on the detail in the software models of processors and ASICs.

With SPILS, the program object code can be developed without waiting for actual hardware and it can be developed in parallel with development of a controller. However, SPILS needs processor models and hardware models that can provide sufficient accuracy and simulation speed. To facilitate wider use of SPILS models, additional model requirements need to be satisfied and these are discussed in Section 3.

2.5. Implementation

The object code is implemented on the hardware that will eventually comprise the target system. In this refinement step, the object code is tested using an actual target system that includes the controller, sensors/actuators and the actual plant system. Furthermore, the object code runs on the target hardware in real time.

Table 2.1 Simulation and design objectives

Name of simulation	Design objective(s)	
MILS	Controller algorithms	
SILS	High-level source code for the controller	
SPILS	Program 'object code' development for a target processor	Software model representing hardware
PILS	Program 'object code' development for a target processor	Hardware
HILS	Program 'object code' implementation and testing using target processor and plant models	Hardware

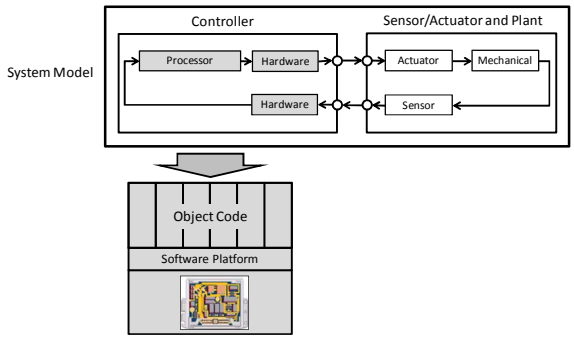

Fig. 2.5 Simulation model for HILS

The simulation used in this design step is called HILS (Hardware In the Loop Simulation). Fig. 2.5 shows a system model for HILS.

In summary, we tabulate the various simulation models and the associated design objectives in Table 2.1.

3. DISCUSSION ON SPILS

The earlier sections highlight the several benefits in using software models that represent the target implementation platform in the ECU development flow. This is also independently confirmed by researchers and developers of embedded software [10]. In this section, we discuss the various issues that need to be addressed to facilitate greater use of software models for ECU processor platforms. We expect that such a discussion will help develop strategies to address these issues and promote wider development and use of SPILS models.

3.1. Simulation speed and accuracy

Based on the abstraction level, the SPILS component models can be developed at various abstraction levels such as:

- Instruction Set Level (ISL)
- Transaction Level Model (TLM) [8]
- Register Transfer Level (RTL)

These SPILS component models tradeoff simulation speed and simulation accuracy. By accuracy we refer to the fidelity with which they model the physical ECU processor platform. For example, RTL models capture a lot of implementation details to facilitate system synthesis and consequently the simulations at the system level would be prohibitively slow. In contrast ISL models are several orders of magnitude faster as they only need to emulate the target processor instruction behavior using the host machine. As a result, the various abstraction levels for modeling, the number of target ECU platforms that are available in the market and the variety of simulation tools available at each abstraction level all together contribute several degrees of freedom. This is graphically illustrated in Fig. 3.1 along with the associated accuracy-performance tradeoffs at each level.

3.2. Development cost

The drawback with the flexibility afforded by such

a large space constituted by the various modeling techniques and simulation tools is that they often do not facilitate the model reuse or model interoperability across toolsets and simulators. This leads to higher costs associated with the use of the models developed. We also discuss other factors that increase the cost of using models in ECU design and development.

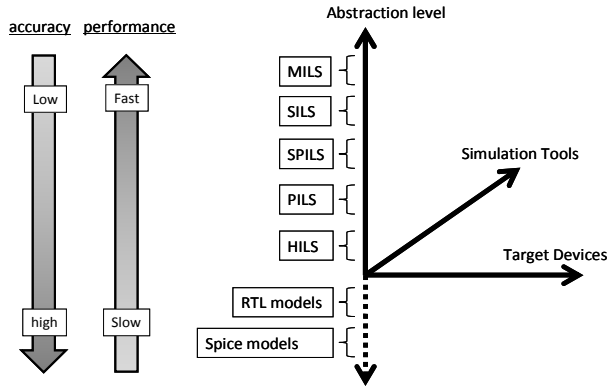

Fig. 3.1 Model design space

Many platform vendors do not develop nor provide higher level models (ISL, TLM) required for system level simulation. The design and development of these models from scratch increases the cost of using these models in design. Moreover, the composition methodology of the language used to describe the models may limit model reuse [9] and the language syntax may make it impossible to mix functional and structural descriptions in system simulations – a requirement to facilitate system evaluation and progressive refinement of the overall system.

A related issue that affects the cost of using models is that, despite the availability of several simulation platforms [4,5] to run software models, the model descriptions are not completely standardized. The ad-hoc designs of model interfaces prevent models from being able to communicate with each other, unless they were originally designed to communicate explicitly.

4. RESOLVING ISSUES WITH SPILS

To address the issue described above we propose to create an ECU model exchange market. The purpose of the ECU model exchange market is to provide an environment to develop and reuse models in ECU design - mainly SPILS in MBD. With sufficient models available, the ECU designers will be encouraged to design with SPILS, thus motivating model developers to develop and provide models. This can be achieved by addressing the issues raised earlier and focusing research on the same. Some directions are discussed below.

- Standardizing model development and defining model interfaces to facilitate model reuse

Standardizing model descriptions will allow the models from different developers to communicate and also be interoperable across simulation platforms from different vendors [3,4]. Interoperable models help reduce the costs otherwise associated with developing models for each simulation platform.

Standardizing the model control interfaces will allow models from different vendors to communicate with each other [9], thereby increasing model reuse. This leads to lower model development costs and encourages model development.

- Benchmarks for model evaluation

The lack of modeling standards, coupled with the variety of simulation libraries and platforms that require modeling, result in several degrees of freedom available for model implementation. Consequently, there is a need for mechanisms to evaluate the models developed against the hardware they model, in terms of their accuracy and the simulation speed they afford. We propose to create a benchmark suite consisting of applications that are representative of the control algorithms that ECUs typically implement. Model evaluation metrics can then be used to quantitatively evaluate the models developed, thereby increasing their utility in the design process.

- Classification of model use-cases

Application dependant performance specifications and implementation constraints result in the availability of as many models as the specifications and constraints themselves. By standardizing the design objectives and requirements for the components models into a few strict classes we can minimize the number of redundant component models.

The ability to parameterize the module instances to customize their behavior also helps reduce the number of redundant component models.

- Security and Intellectual property (IP)

Accurate modeling often requires capturing device specific details in the component models. These details may often be proprietary in nature and the models would need to be secure to protect the IP of the respective owners to encourage sharing and reuse of these models

Making the ECU model exchange market work relies on the support of embedded system manufactures, ECU suppliers, semiconductor companies, and tool vendors. We expect that the ECU model exchange market will be beneficial to all the stakeholders as it will help accelerate ECU development and result in the development of competitive embedded products. It will also propel the development of semiconductor products and tools for ECU.

5. RELATED WORK

The objectives behind the UNISIM framework [3]

most closely resemble the position we take in this paper. They promote modular software development, model interoperability, multi-level abstractions for models, and advocate an open library/repository for the simulation modules. Their platform is built on top of the SystemC [6,7,8] framework. We reiterate their objects and motivate further research in model development and evaluation.

Recently, several commercial vendors have also begun to produce software simulation models that claim higher model accuracy and faster simulation speed [4,5]. Their presence reiterates our position and strengthens the need to address the requirements highlighted in this paper.

6. FUTURE WORK

Design methodologies that use simulators to substitute for actual hardware is important to shorten the time to market and decrease design costs in embedded system design

For simulators to be effective, it is necessary to have appropriate models used in embedded systems. However, the models are not easy to develop and their availability stays low due to lack of standardization for model specification and model description. This makes it difficult to re-use the models resulting in higher model development costs.

To address the issues we propose standardization of the model specifications, description and interface definitions to lead to increased model interchange.

REFERENCES

[1] Edwards S., Lavagno L., Lee E.A., Sangiovanni-Vincentelli A., "Design of embedded systems: formal models, validation, and synthesis", Proceedings of the IEEE, Volume 85, Issue 3, Mar 1997, pp.366-390

[2] Henrik Lonn and Ulrich Freund, "Automotive Architecture Description Languages", Automotive Embedded Systems Handbook, CRC Press, 2009.

[3] August D., Chang J., Girbal S., Gracia-Perez D., Mouchard G., Penry D., Temam O., Vachharajani N., "UNISIM: An Open Simulation Environment and Library for Complex Architecture Design and Collaborative Development", Computer Architecture Letters, July-December 2007, Volume 6, Issue 2, pp. 45-48

[4] CoWare Web Site, "http://www.coware.com/"

[5] VaST Systems Technology Web Site, "http://www.vastsystems.com/"

[6] SystemC, OSC Initiative. OSCI Tech. Rep., 2003.

[7] P.R. Panda, "SystemC - A modeling platform supporting multiple design abstrations", ISSS'01.

[8] Grotker et.al., System Design with SystemC, Kluwer Academic, 2002.

[9] M. Vachharajani et.al., "Microarchitectural Exploration with Liberty," Proceedings MICRO, 2002.

[10] H. Shokry and M. Hinchey, "Model-Based Verification of Embedded Software". IEEE Computer, April 2009.