A Prototyping Environment for Model-Based Codesign

S. Schulz and J.W. Rozenblit
Dept. of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721-0104
USA

{sschulz|jr}@ece.arizona.edu

K. Buchenrieder and M. Mrva
Siemens AG
ZT ME 5
Otto-Hahn-Ring 6
81739 München
Germany
{buchen|michael.mrva}@mchp.siemens.de

Abstract

In this paper, a prototyping environment for embedded systems design is presented. This environment supports a design methodology called model-based codesign and fosters implementation independent specification of embedded systems. In this article, the focus is on the technology allocation and implementation phases of the methodology.

1. Introduction

Recent advances in embedded systems have resulted in rapid emergence of products such as global positioning systems, efficient electronic fuel injection, portable CD players or palm top computers. Stringent processing and reliability requirements imposed on these complex systems as well as the need for low cost and little time-to-market make a revision of the traditional design methods necessary.

A number of authors [2,3,7,8,13] have been suggesting various approaches to address these issues using hardware/software codesign. We have been proposing a high level system design approach called model-based codesign that relies heavily on simulation modeling techniques to explore the feasibility of virtual prototypes [1,12]. It provides a methodology for the design of embedded systems which fosters rapid development, late partitioning and assessment prior to implementation, as well as model and hardware/software module reuse.

In [12], we have presented the foundations of our design approach and demonstrated it using an automotive safety control device. This paper introduces a protoyping environment used to realize the theory-based concepts.

2. Model-Based Codesign

Formal specification techniques are of limited effectiveness without a systematic modeling methodology guiding their use throughout the codesign process. Therefore an approach to hardware/software codesign is proposed [12] that uses stepwise refinement of models and facilitates component specifications at multiple levels of abstraction. The system under development can then be tested according to requirements and specifications using simulation. No commitments regarding an implementation in hardware or software are made early in the design process.

The design procedure is described in detail in [1] and [12]. Initially, requirements and specifications are obtained for the system to be modeled. The system is then described as an abstract model which is a combination of its structural [10], functional, and associated behavioral [4] facets. It is emphasized that the model components are described at a high level of abstraction to remain technology independent. The modeling process is accompanied by a stepwise refinement of the model down to a desired level of granularity.

In model-based codesign, *experimental frames* can be employed [9] to verify the model's correctness during simulation. These frames specify conditions under which the system is to be observed. At the end of the iterative simulation process, a virtual system design is obtained.

A critical phase of in our design methodology is the so-called *model mapping*. In model mapping, the simulatable model descriptions are translated into hardware, software, and interface specifications. The basis for the model mapping is the selection of an underlying architecture platform which determines the execution time of software components. A search through the design space results in a final implementation for the verified model.

3. Model Mapping

Model mapping is preceded by an extensive modeling, refinement and simulation phases [12]. To obtain a simulatable specification, we use the Discrete Event System Specification formalism (DEVS). The DEVS formalism introduced by Zeigler [13] provides a means of specifying a

mathematical object called a system and supports building models in a hierarchical, modular manner. We arrive at a final verified model after an iterative modeling and simulation loop performed in DEVS-Java, an object-oriented implementation of DEVS. The product of all the necessary refinement and modification is referred to as the virtual board VB* which satisfies all the functional requirements. Now, the components of the verified model, i.e., a virtual prototype of the system, have to be assigned to either hardware or software processes, and properly interfaced with each other as shown in Figure 1. The mapping of the application to a demonstrator is done by partitioning the system at the component level.

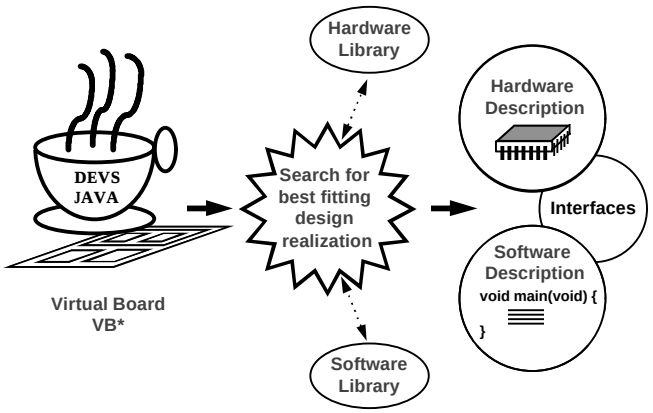

Figure 1. Search for Best Fitting Design

The partitioning of VB* can be done either interactively by the designer or can be supported by an algorithm that searches the design space for the best fitting implementation. Due to the different levels of granularity of the model components the mapping is also allowed at different levels of abstraction with respect to their corresponding hardware or software modules (i.e., descriptions at register transfer, gate or chip level for hardware components and instructions, functions or programs for software components).

The assignment process begins with the selection of a hardware platform for VB* to run software processes on. This information needs to be known for the search of the best fitting design since the architecture specifics, e.g., instruction set or processing speed, define important run time parameters for the software modules. The use of libraries containing hardware and software modules fosters reuse of already implemented, tested, and verified designs. For future development, it will also be easier to keep up with advances in technology that might change the way a particular hardware or software element is implemented.

Other requirements that help limit the search space are hard implementation constraints or non-functional requirements given for the system. For example, only a certain chip could be available for a part of the design which would limit the choice of bus controller to this specific chip. The rest of the assignment will the be mostly guided by timing constraints known from the model simulation. Component by component, from the bottom level to top level VB* is then built by verifying real-time timing constraints of the composed. If there is a request for an impossible model mapping for a specified design the trouble spot can be identified during the mapping to indicate where the constraints for the implementation need to be modified. After that we have to repeat our simulation experiments to verify our modified design.

The mapping of VB* onto the demonstrator is done manually at this stage of the project. Factors influencing the assignment process are realizability, speedup, and cost of the implementation. Hardware is used to accelerate the execution time of the process body for some components. The final design implementation then consists of a complete description of a system to be designed preferably using hardware and software description languages. For the implementation of the VB*, this description is synthesized and compiled by tools that are commercially available on the market.

Interfacing is needed to have the different components or processes communicate with each other. The three major categories of interfacing are: hardware with hardware, software with software, and mixed hardware with software. Choosing the appropriate interface ensures correct program execution, handles synchronization and protects shared variables in the case of parallel execution of components. Techniques used in the implementation of interfaces include interrupt handling, handshake protocols, semaphores, and busy waiting, etc.

4. The Protoyping Environment

An overview of the prototyping environment is shown in Figure 2. The reference architecture for implementation of the software components is a Motorola MC68HC11 microcontroller. This microcontroller was chosen since it is commonly used by the automotive industry for engine control applications. The hardware functions generated during the model mapping are realized on an ALTERA MAX9320 Field Programmable Gate Array (FPGA). The synthesis tool for this chip can either process VHDL descriptions or graphic design entry to create desired circuits. The two chips are interfaced via a 16-bit communication bus. Due to the limited availability of I/O pins, data transfers are multiplexed to increase the communication bandwidth.

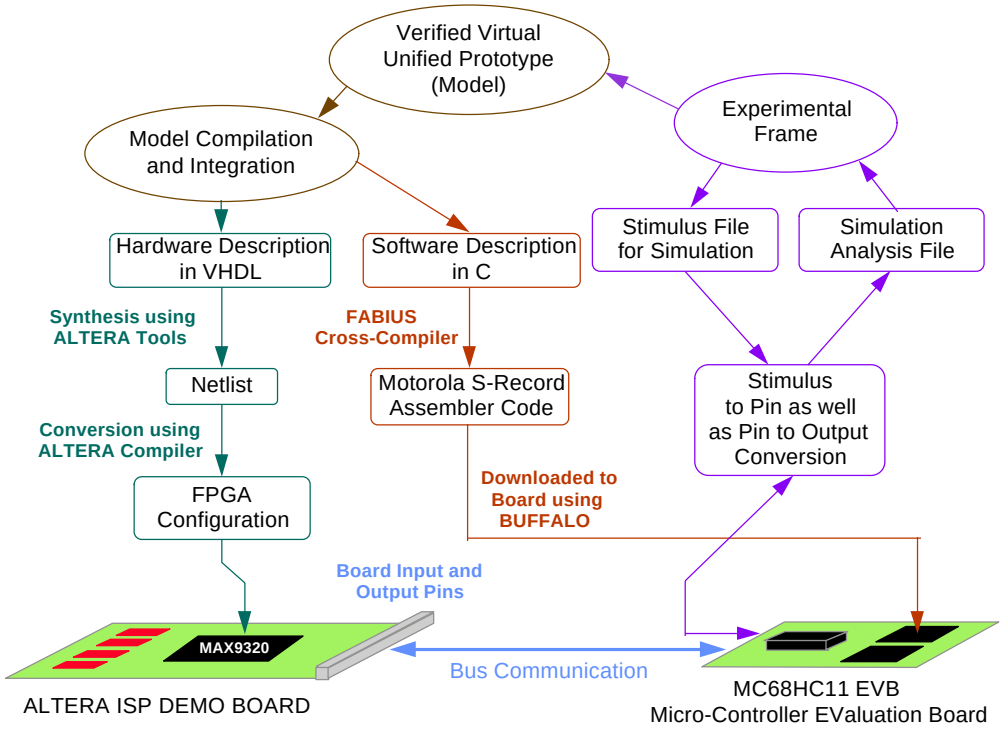

Figure 2. Prototyping Environment for Model Testing and Physical Realization

The implemented model components are interfaced by the Simple Modeling Operating System (SMOS). This operating system is based on a centralized dynamic scheduling approach. For our purpose, we define modeling components to be processes. A process consists of a process body, mailbox, and carries certain properties, for example: priority level, number of instructions, etc. Processes are placed on a waiting list by an interrupt service routine or

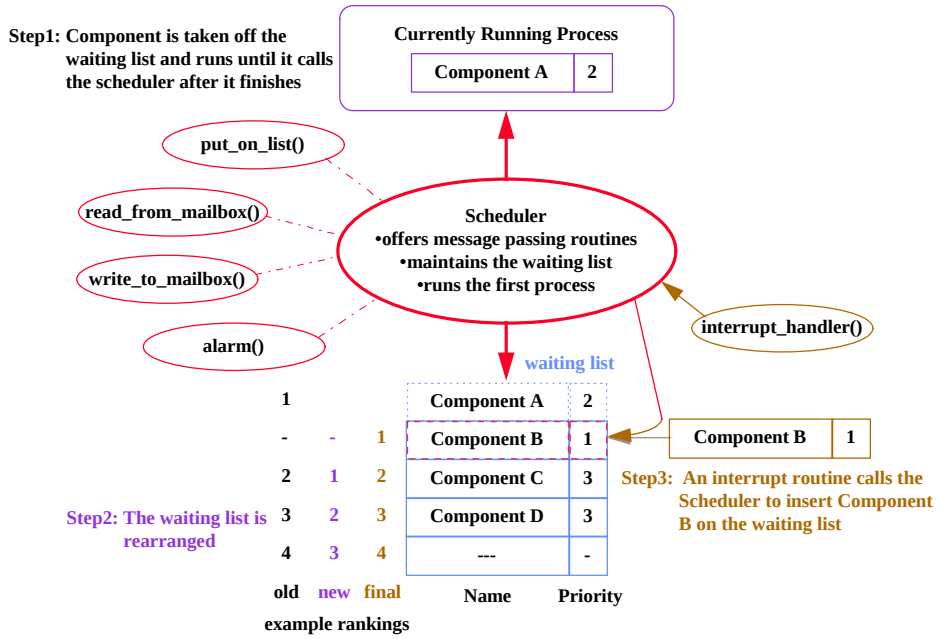

Figure 3. Operation of SMOS

other components using a priority scheduling scheme. Inter-process communication is achieved by message passing. SMOS is preemptive in the sense that it will remove components running past a certain time-out limit. That component will be stopped by means of a timer interrupt and an alarm is issued. The process is then killed and the user is notified of a malfunctioning component. Figure 3 illustrates the features described above.

The execution of a model component is initiated either by the currently running process or by incoming data. These data can be send again by the running process or from external sources, e.g., vehicle sensors or the driver via interrupt. An interrupt handler puts the data into the SMOS format. When the component finally gets its turn as the running process, it first sends out its process number to the hardware port which will select corresponding supporting hardware function(s). The software body of the component then obtains values necessary for execution of the process body from its mailbox, executes, sends out results to (an)other component(s), and finally schedules (an)other component(s) to be placed on the SMOS waiting list. The process body can be either implemented as a software routine, a hardware function or a hybrid where some hardware functionality supports and accelerates a software routine.

5. The Application: Autonomous Intelligent Cruise Control

As a sample study we considered a design from the automotive domain. Codesign in general lends itself well to the design of the new intelligent generation of electronic automotive applications [5]. There, the car industry faces reduced design cycles and is keen on saving costs. At the same time, the electronic components used in automobiles become highly complex embedded systems. The traditional, separate design of hardware and software and software components is difficult to justify, especially in view of the sophistication of the functional characteristics of the components being built.

One such application which has recently received a lot of attention is a device called autonomous, intelligent cruise control unit (AICC). The AICC system can be understood as an extension of the regular cruise control, not only keeping a fixed speed, but also adapting to the speed of the vehicle ahead. It controls the relative speed between two vehicles traveling in the same lane. Furthermore, it asserts longitudinal elements of control but no lateral control. Although the system is autonomous, meaning it does not rely on communication between vehicles, the driver remains in full control since the driver can override the device, e.g., by braking. For our purposes we require the unit to keep the speed of the vehicle within a margin of +/- 2 km/h of the safe speed or set speed - depending which one of the two speeds is lower. The safe speed is defined as the maximum speed that would be allowed to keep the car within a safe distance to the vehicle ahead. The set speed is the speed requested by the driver.

The circuitry of a safety unit must satisfy real-time constraints. It should not fail in emergency situations. This includes returning the control to the driver in case of a failure of the system to not make matters worse. A standard cruise control nowadays does not represent a true real-time design problem. However, the design of an AICC system must take into account large amounts of data, especially from the vision sensors, and process such data in a timely manner. All the safety requirements have to be met within a small margin of error under normal traffic conditions.

Our design example focuses on the AICC control unit which is the main processing element in such a device. For a more detailed description of the modeling of the AICC control unit it and its environment unit we refer the reader to [11].

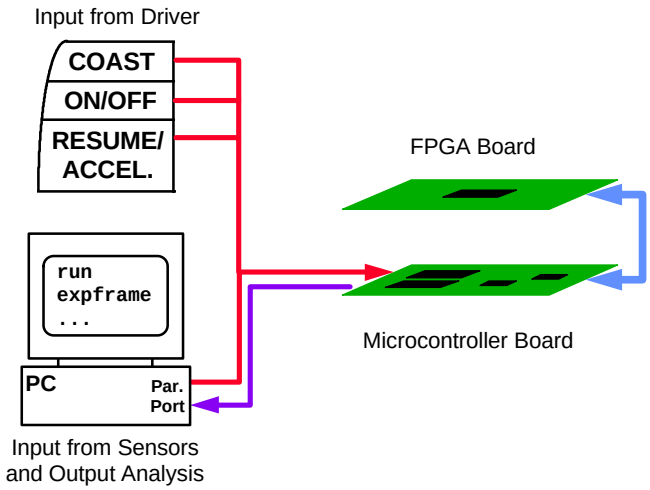

Figure 4. Testing Setup

6. Testing Setup

In order to verify the functionality and correctness of the final demonstrator module a testing environment was built as shown in Figure 4. For that purpose the concept of experimental frames is transferred from the simulation domain to the real-time domain. A C-program reuses the experimental frames used during simulation and generates inputs in real-time. It also obtains outputs from the demonstrator and analyzes the data.

The inputs to the application can be divided into control and data inputs. Some of the control inputs can be realized

using the available push button switches located on the ALTERA ISP Demo Board. Data as well as control inputs are generated by a C program running on the 486 IBM PC. There, data can either be manipulated by user or automatically by the program from the PC and is then sent to the EVB via the parallel I/O port.

The inputs arrive at the microcontroller making use of its external interrupt line. A handshake protocol is used the transfer of the data from the PC. Some interrupt logic had to be designed to eliminate switch debouncing on the ALTERA board and to handle multiple interrupt requests. Some debugging information from the implemented application can also be displayed for evaluation using the serial interface with the PC.

7. Summary

This paper describes our initial efforts to build a prototyping and testing environment to support the model-based codesign framework. The environment provides a unique ability to verify embedded systems' model in the form of formal, simulatable specifications, and to re-evaluate the hardware, software, and interface physical realization in an emulation loop of experimental frames prior to the system's deployment.

Acknowledgments

This work has been supported by Siemens AG, Central Research and Development Laboratories, München, Germany. We would like to thank Mr. Joe Hanson at the ALTERA University Program for providing the ALTERA ISP Demo Board and the corresponding development software.

References

[1] K. Buchenrieder and J.W. Rozenblit, "Codesign: An Overview", in J.W. Rozenblit and K. Buchenrieder (Eds) *Codesign: Computer-Aided Software/Hardware Engineering*, pp. 1-16, IEEE Press, 1994.

[2] R. Ernst, J. Henkel, and T. Benner, "Hardware-Software Cosynthesis for Microcontrollers", *IEEE Design and Test of Computers*, 10(4), pp. 64-73,1993.

[3] R.K. Gupta and G. De Micheli, "Hardware-Software Cosynthesis for Digital Systems", *IEEE Design and Test of Computers*, 10(3), pp. 29-41, 1993.

[4] D. Harel, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", *IEEE Transactions on Software Engineering*, 16(4), pp. 403-14, 1990.

[5] X. Hu and J. D'Ambrosio, "Codesign of Architectures for Automotive Powertrain Modules", *IEEE Micro*, 14(4), pp. 17-24, 1994.

[6] P.A. Ioannou and C.C. Chen, "Autonomous Intelligent Cruise Control", *IEEE Transactions on Vehicular Technology*, 42(4), pp. 657-72, 1993.

[7] A. Kalavade and E.A. Lee, "A Hardware-Software Codesign Methodology for DSP Applications", *IEEE Design and Test Computers*, 10(3), pp. 16-28, 1993.

[8] S. Kumar, *A Unified Representation for Hardware/ Software Codesign*, Ph.D. Dissertation, University of Virginia, 1995.

[9] J.W. Rozenblit, "Experimental Frame Specification Methodology for Hierarchical Simulation Modeling", *International Journal of General Systems*, 19(3), pp. 317-336, 1991.

[10] J. Rumbaugh, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[11] S. Schulz, J.W. Rozenblit, and K. Buchenrieder, "Towards Model-Based Codesign: An Intelligent, Autonomous Cruise Controller Application", *Proceedings of the 1997 IEEE Conference and Workshop on Engineering of Computer Based Systems*, pp. 73-80, Monterey, CA, 1997.

[12] S. Schulz, J.W. Rozenblit, M. Mrva, and K. Buchenrieder, "Model-Based Codesign: the Framework and its Application", to appear in *IEEE Computer*.

[13] D.E. Thomas, J.K. Adams, and H. Schmit, "A Model and Methodology for Hardware-Software Codesign", *IEEE Design and Test of Computers*, 10(3), pp. 6-15, 1993.

[14] B.P. Zeigler, *Object Oriented Simulation with Hierarchical Models*, Copyright by Author, 1995.