A Codesign Environment Supporting Hardware/Software Modeling at Different Levels of Detail

Sanjaya Kumar, Fred Rose
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{skumar, rose}@htc.honeywell.com

Abstract

Hybrid modeling is a technique that integrates performance and behavioral models within a common simulation. This approach allows behavioral components, containing mixtures of hardware and software, to be evaluated within the context of the system being developed. Hybrid interfaces are required to integrate the behavioral models with the performance models.

This paper presents Honeywell's VHDL-based approach to codesign using hybrid modeling. The structure of the hybrid interface is described, and a hybrid interface for a processor model is presented. A four processor Myrinet example is provided to illustrate hardware/ software modeling at different levels of detail. We are evaluating our methodology using an internal application.

Keywords: hybrid modeling; performance evaluation; multi-level modeling.

1. Introduction

Several problems exist in the design of complex systems. Currently, there is a separation between those who develop the system architecture and those who create the detailed hardware/software implementation for the system. This separation leads to the model continuity problem [1], the inability to refine a system level model into a hardware/ software implementation. Another problem is the failure to appreciate the subtleties associated with integrating subsystems. For example, the 90/50 rule [2][3] states that although 90 percent of ASICs work the first time, only 50 percent work properly in the system. Also, the increasing complexity of systems mandates the use of methodologies and techniques which support design and analysis at multiple levels of detail.

Hybrid modeling [4][5], the simulation of performance (uninterpreted) models and behavioral (interpreted) models in a common environment, attempts to address these problems. This technique allows behavioral components, containing mixtures of hardware and software, to be evaluated within the context of the system being developed. Hybrid interfaces are required to integrate the behavioral models with the performance models.

This paper presents Honeywell's VHDL-based approach to codesign using hybrid modeling. The structure of the hybrid interface is described. A hybrid interface for a processor, an example of a library element within the Hybrid Model Library (HML), is discussed. The HML architectures are hybrid versions of various performance modeling constructs contained in the Honeywell Performance Model Library (PML), a library of VHDL-based modules used for performance analysis that employ tokens. A four processor Myrinet hybrid model is provided to illustrate multi-level hardware/software modeling.

We are in the process of evaluating our hybrid modeling methodology using an internal Honeywell application. The objectives are to better understand the benefits of hybrid modeling, beyond those discussed earlier, and to evaluate our current approach.

The remaining portions of the paper are organized as follows. Section 2 describes the PML. Section 3 provides background material. Section 4 discusses the hybrid interface. Section 5 presents a processor hybrid interface. A hybrid modeling example is provided in Section 6. Section 7 summarizes the contributions of the work.

2. The Performance Model Library

The Honeywell PML [6][7] was created to fill the analysis needs required for the design of large, distributed, embedded real-time systems. This library is being

0-8186-7895-X/97 $10.00 © 1997 IEEE

incorporated into the commercial Performance Modeling Workbench (PMW) [8] product being developed by Omniview, Inc. Design features of this library include processing elements, communication components (routers, crossbars, etc.), system input/output and storage, topology, software partitioning, allocation, and scheduling.

The PML utilizes standard commercial VHDL capabilities. As a result, the library allows a system architect to capture the system under study in a consistent, verifiable form. Standard output routines tabulate and graph performance statistics such as utilization and latency.

Three primary classes of library elements are used in the PML: the leaf cells, the communication cells (e.g. Myrinet components [9]), and the processor model. All of these modules communicate through the use of tokens [10].

The processor model facilitates hardware/software codesign and coanalysis [11][12]. It consists of three parts: software models or tasks, a scheduler or thread manager, and the hardware core. The processor model provides a powerful software modeling capability over a wide range of modeling levels. Additional features have been added to handle interrupts, preemptive tasking, task communication, task synchronization, and other services. The scheduling model supports static and dynamic tasking, and rate monotonic scheduling.

3. Hybrid Modeling Background

3.1 Hybrid Model Attributes

The technique for developing hybrid models depends on the class of modeling problems being solved. The classes of hybrid modeling are defined by those model attributes which fundamentally alter the implementation of the hybrid interface [5]. These attributes include: the hybrid model objective (timing vs. functional verification), the timing and synchronization mechanism across the hybrid interface, the type of interpreted model (e.g. hardware, hardware/software), the data transformation mode (i.e, the technique used to generate valued data for the interpreted model), and the data type of the interpreted signals (e.g. bit vectors or integers).

3.2 Interfacing Scenarios

The interfacing scenarios [5] define the data flow between the uninterpreted and the interpreted domains. Some scenarios translate data across a single boundary (U/I, I/U), and others translate data from one domain and then back (U/I/U, I/U/I). The U/I and U/I/U scenarios are illustrated in Figure 1.

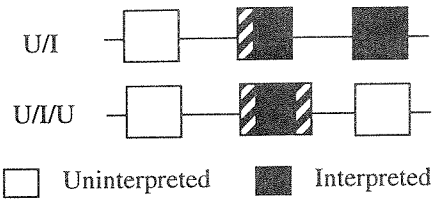

Figure 1. Two Interfacing Scenarios [5]

3.3 Relation to Codesign Process

Hybrid modeling fits nicely within the context of the Rapid Prototyping of Application Specific Signal Processors (RASSP) codesign process [13][14], as defined by Lockheed Martin Advanced Technology Laboratories. The major phases include system definition, architecture definition, and detailed hardware/software design.

Performance modeling is often used to verify that the proposed architecture will meet performance requirements. Within the overall system architecture, this analysis could occur during the system definition phase. For smaller sections of the architecture, this analysis will occur during the architecture selection and verification phase.

As the design evolves, it may be desirable or perhaps necessary to consider detailed design alternatives in order to minimize risk. Following the spiral model of development [15], this results in the spawning of mini-spirals to develop high risk items. Based upon risk, portions of the design may be at differing maturity levels.

At this point, hybrid modeling becomes important. As the system model is refined, certain high risk components can be developed further. Specifically, certain components within the system level uninterpreted model can be replaced with more detailed interpreted models, supporting hardware/software modeling at various levels of detail.

4. HML Hybrid Interface

To illustrate the use of hybrid interfaces, one possible application of these interfaces is shown in Figure 2. In this example, one performance model (white box) is interacting with two behavioral models (black boxes) using hybrid interfaces (striped boxes). The hybrid interface to the bus is a token-based interface, and the hybrid interface to the behavioral model is a valued interface consisting of, for example, bits.

In the HML, a hybrid interface consists of three logical components. The Uninterpreted Hybrid Interface (UHI) interacts with tokens and returns data back to the performance model. The Hybrid Interface Core (HIC) handles data translation (for example, from "tokens" to bit vectors), abstraction, and generation, as well as timing and

synchronization between the UHI and IHI. It accepts data from the IHI and gives it to the UHI. The Interpreted Hybrid Interface (IHI) interacts with the interpreted domain. The IHI is the most complicated component since the interpreted domain must potentially represent many different abstractions, forms, and types. The IHI also handles timing and synchronization specific to the behavioral component. In some cases, it performs clock generation for the behavioral component.

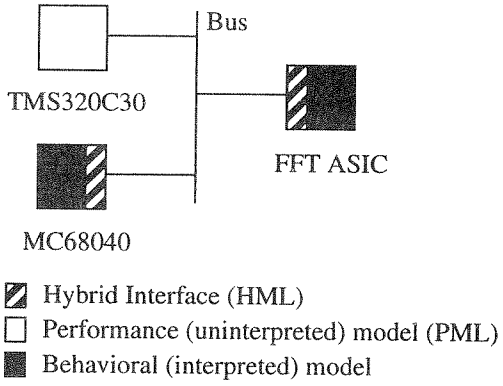

Figure 2. Hybrid Interface Example

Hybrid interfaces have standard port and generic interfaces. The port interface is the PML entity so that a PML element will have one entity, and at least two architectures: performance and hybrid. This allows easy insertion of hybrid elements into a PML design.

The VHDL implementation of a typical hybrid interface consists of a process and three components: a queue, the IHI, and a global data store (see Figure 3). This example corresponds to the iodevice module (a leaf cell), which can be used to model the performance aspects of a memory. The UHI and HIC are contained within the process of the hybrid architecture. The UHI/HIC, global data store, and queue components are taken from the HML and the PML. The components shown in bold will differ between memories and are user-specified. In this example, a RAM is used as the interpreted component.

Three "standard" signals are utilized to interface the UHI/HIC with the IHI. The *Intftoken* signal carries a request token that contains the operation to be performed by the interpreted component. In the case of the RAM, the operation would be either a read or a write. The *IOPort* signal connects to the global data store. The HIC writes translated (or generated) data values, such as address and data in the case of the RAM, into the global data store. These values are read by the ICI, converted into a format appropriate for the RAM, and applied to the RAM. The *Retdata* signal is used to return information back to the performance model, for example, during a read operation. The returned data is incorporated into a token and sent to other PML modules via the output queue.

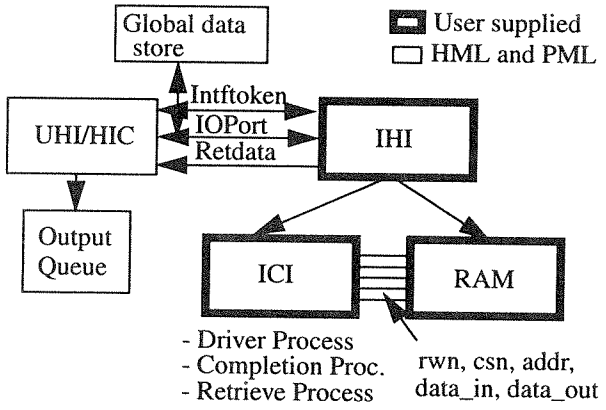

Figure 3. Interfaces within a Hybrid Architecture

The ICI is an important element. It consists of the driver process, the completion process, and the retrieve process, which apply the appropriate values to the interpreted component, check to see if the operation is complete, and return any data back to the HIC, respectively.

5. Processor Hybrid Interface

The processor hybrid interface may be used with either a processor element containing memory and even possibly a switch device, or a custom computing device. The current hybrid architecture was developed with a simple interpreted component consisting of an instruction set level cpu, a memory, and a bus communication element.

The hybrid processor architecture is illustrated in Figure 4. A hybrid token watcher, not part of the hybrid interface, accepts tokens from *HWport* (a bus interface). The UHI and HIC sections are similar to the corresponding sections in other hybrid architectures. The processor hybrid interface accepts read requests, write requests, and requests for executing programs in memory. The primary difference between this model compared to others is that the interface has requests coming from both the uninterpreted and interpreted domains. For example, the interpreted component may want to send a message to another processor. Thus, an additional signal *Extrequest* is required to support generating a token to the uninterpreted domain.

The "watch-and-react" interface [16] is an alternative processor hybrid interface consisting of triggers and drivers. The trigger is used to detect events or changes in primary variables, and the driver is used to apply values to its probes, which represent signals of interest within the interpreted model.

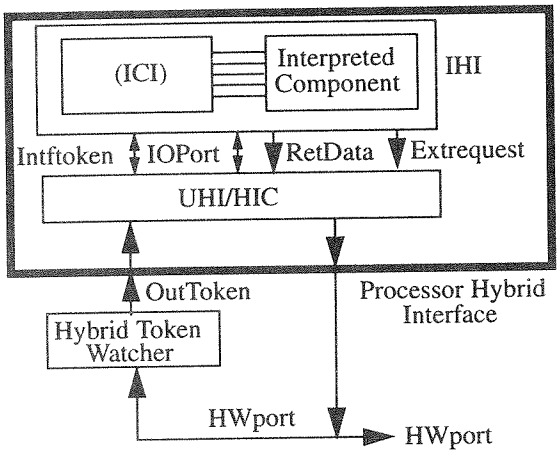

Figure 4. Hybrid Processor Architecture

6. Four Processor Myrinet Hybrid Model

To illustrate the hybrid modeling capability, a four processor Myrinet example is presented. The model (see Figure 5) consists of four processors, four Myrinet network interfaces (NIFs), and a four port Myrinet switch. The components in bold correspond to hybrid elements.

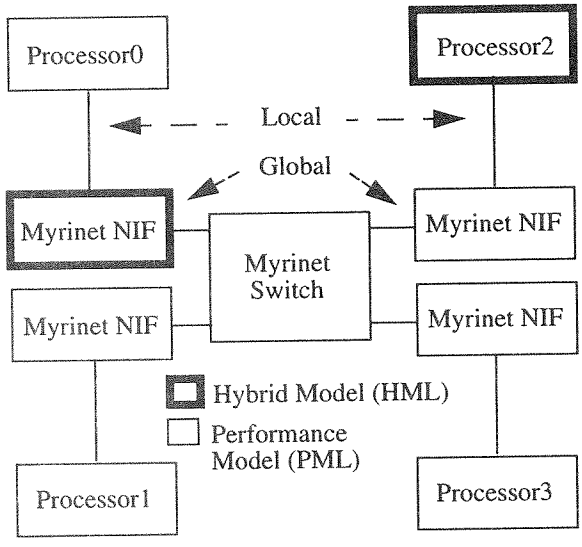

Figure 5. A Four Processor Myrinet Example

The processor blocks represent software descriptions executing on hardware models at varying levels of detail. The PML processor elements contain periodic tasks, a scheduler, and a Pentium hardware model. The tasks can be described in terms of scheduler operations, for example, "post an event," virtual instructions, such as context switch, or processor instructions, such as floating point divide. The tasks in the example communicate by sending and receiving events. The hybrid processor model (Processor 2) utilizes the hybrid interface discussed in Section 5. It consists of an algorithm described using actual instructions that are fetched and executed by a simple instruction set level processor. Thus, the PML and HML processor models are described at two different levels of detail.

The arrival of a token from Processor0 on the local port triggers the execution of the hybrid Myrinet NIF architecture. The sending and receiving of messages are performed by the interpreted component (an abstraction of the actual Myrinet NIF device) upon the arrival of a request token on the *Intftoken* signal within the IHI (see Figure 3). The generated address and data are written into a global data store by the HIC, read by the ICI, and then applied to the interpreted component as mentioned in Section 4. Details regarding the hybrid Myrinet NIF operation can be found in [17].

As illustrated in Figure 6, the interpreted component in the hybrid processor model consists of an instruction set level cpu, a memory, chip select logic (not shown), an abstract bus interface unit, and a clock. The cpu executes the program stored in the memory. Chip select logic is used to enable the memory or the bus interface unit. The bus interface unit mediates the flow of information into and out of the cpu/memory. For example, upon receiving data from the ICI, it interrupts the cpu.

Similar to the hybrid Myrinet NIF, the arrival of a request token within the IHI initiates the execution of a driver process within the ICI. The driver determines which operation is to be performed, reads data from the global data store, and applies the correct values to the interpreted component through the interfacing signals.

The ICI responds to three types of requests from the performance model. A READTOKEN request is used to indicate that an internal location within the interpreted component's memory is to be read. A WRITETOKEN request indicates that an internal location within the interpreted component's memory is to be written. A CONTROLTOKEN request indicates that a specific program within the interpreted component is to be executed. Parameters associated with these requests are populated within the request token. Once such a request is received, the appropriate signal values are applied to the interpreted component through the abstract bus interface unit. The abstract bus interface unit signals the ICI when the operation is complete and returns data to the ICI when a read is requested.

The hybrid processor model is an example of a U/I scenario. However, it can also be viewed as a U/I/U scenario since information flow occurs in both directions (U/I and I/U).

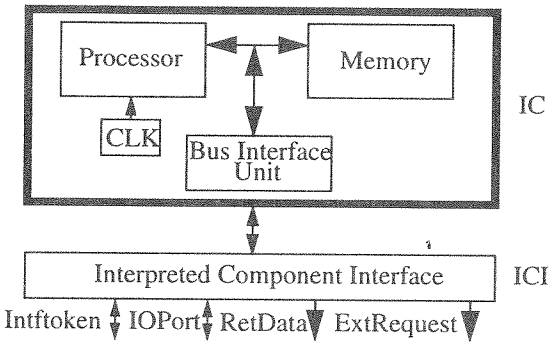

Figure 6. Interpreted Component

7. Conclusions

The contributions of this work focus on providing a capability that supports the hybrid modeling of complex systems. This capability allows hardware/software trade-offs to be evaluated at multiple levels of detail. It also supports some level of model continuity and allows hardware/software sub-systems to be analyzed within the context of the system being developed.

We have developed a modeling structure to serve as a basis for constructing hybrid interfaces. Also, we have developed a library of hybrid interfaces, and have presented one in particular, the processor hybrid interface. Although not mentioned due to space limitations, we have also created the Hybrid Interface Generation (HIG) toolkit [17] to aid in the construction of hybrid interfaces. We are in the process of further evaluating our methodology.

Acknowledgments

The HML was developed under contract # F33615-94-C-1495 from Wright Labs, Dayton OH.

The PML was originally developed under contract # F33615-92-C-3802 from Wright Labs and has been significantly enhanced under the RASSP program. The RASSP program is a four and one-half year, $150 million Defense Advanced Research Projects Agency (DARPA)/ Tri-Service initiative intended to dramatically improve the process by which complex digital systems, particularly embedded digital signal processors, are designed, manufactured, upgraded, and supported. Under RASSP, PML work has been done under the following contracts: DAAL01-93-C-3380 and F33615-94-C-1495. HML development was also sponsored by the RASSP program.

Todd Carpenter, John Shackleton, and Todd Steeves, in addition to playing key roles on the PML development, have contributed ideas to the hybrid model library. Sabera Kazi has contributed to the hybrid model library development and verification. Minesh Amin was the principal author of the Hybrid Interface Generation toolkit.

References

[1] Franke, D. W., M. K. Purvis, "Hardware/Software Codesign: A Perspective," *Proceedings of the 13th International Conference on Software Engineering*, May 13-16, 1991, pp. 344-352.

[2] Bourbon, B., "On System Level Design," *Computer Design*, December 1990.

[3] Schultz, S. E., "An Overview of System Design," *ASIC & EDA*, January 1993, pp. 12-21.

[4] Aylor, J. H., R. Waxman, B. W. Johnson, R. D. Williams, "The Integration of Performance and Functional Modeling in VHDL" in *Performance and Fault Modeling with VHDL*, J. Schoen, ed., Prentice-Hall, Englewood Cliffs, NJ, 1992.

[5] Meyassed, M., R. McGraw, J. Aylor, R. Klenke, R. Williams, F. Rose, and J. Shackleton, "A Framework for the Development of Hybrid Models," *Proceedings 2nd Annual RASSP Conference*, Arlington, VA, July, 1995, pp. 147-154.

[6] Rose, F., T. Steeves, and T. Carpenter, "VHDL Performance Models," *Proceedings 1st Annual RASSP Conference*, Arlington, VA, August, 1994, pp. 60-70.

[7] Steeves, T., F. Rose, T. Carpenter, J. Shackleton, O. von der Hoff, "Evaluating Distributed Multiprocessor Designs," *Proceedings 2nd Annual RASSP Conference*, Arlington, VA, July, 1995, pp. 95-102.

[8] Performance Modeling Workbench User's Manual, Version 0.9, Omniview, Inc., Pittsburgh, PA, 1996.

[9] Boden, N. J., et al., "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, February 1995, pp. 29-36.

[10] Honeywell Technology Center, VHDL Performance Modeling Interoperability Guideline, Version 1.6, November, 1995.

[11] Rose, F., T. Carpenter, S. Kumar, J. Shackleton, and T. Steeves, "A Model for the Coanalysis of Hardware and Software Architectures," *Proceedings of the 4th International Workshop on Hardware/Software Codesign*, March 18-20, 1996, Pittsburgh, Pennsylvania, pp. 94-103.

[12] Kumar, S., J. H. Aylor, B. W. Johnson, W. A. Wulf, *The Codesign of Embedded Systems - A Unified Hardware/ Software Representation*, Kluwer Academic Publishers, Boston, Massachusetts, 1996.

[13] Richards, M., "The RASSP Program Overview and Accomplishments," *Proceedings 1st Annual RASSP Conference*, Arlington, VA., August 1994, pp. 1-8.

[14] Saultz, J.,"Lockheed Martin Advanced Technology Laboratories RASSP Second Year Overview," *Proceedings 2nd Annual RASSP Conference*, Arlington, VA, July, 1995, pp 19-31.

[15] Boehm, B. W., "A Spiral Model of Software Development and Enhancement," IEEE Computer, May 1988, pp. 61-72.

[16] Dungan, W. W., R. H. Klenke, J. H. Aylor, "A Watch-and-React Interface for Hybrid Modeling," Department of Electrical Engineering, University of Virginia, Technical Report No. 960531.0.

[17] Kumar, S., F. Rose, "Integrated Simulation of Performance and Behavioral Models," *Proceedings of the Fall '96 VIUF Conference*, Durham, NC, October 27-30 1996, pp. 185-194.