A Framework for the Development of Hybrid Models

Moshe Meyassed, Robert McGraw, James Aylor, Robert Klenke, Ronald Williams
University of Virginia

Fred Rose, John Shackleton
Honeywell Technology Center

Abstract

Rapid prototyping of complex digital systems requires a well defined design flow. A typical top-down design flow starts with a construction of a performance model of the system under design, which helps in making architectural design decisions. Unless this model is used for later phases of the design process, a model continuity problem exists. This problem results from having to model and simulate systems using different design environments for different levels of design detail. Most of the levels of the design process do not exhibit the model continuity problem. However, this problem is prevalent between the performance and functional modeling levels. The work presented here allows for the true step-wise refinement of a performance (uninterpreted) model into a functional or behavioral (interpreted) model. The critical hurdle to the realization of this methodology is the ability to do hybrid modeling. Hybrid modeling is the capability of mixing high-level performance constructs and functional components in a common analysis environment. Hybrid modeling supports the model refinement design flow by providing an interface to bridge the information gap between performance models and behavioral implementations.

1. Introduction

The goal of automating the process of digital design has produced many tools and methodologies for each stage of the design process, from requirements generation through fabrication. However, each methodology has concentrated on a particular design task, and this concentration has led to the fragmentation of the design process. Each design group develops models of the system at a level of detail appropriate to the group's design objective. Traditionally, these models take a variety of forms in a variety of languages. Each model requires different CAD tools for the simulation and analysis of the design. As a result, the design teams generate several different models of the system which may not interact with each other. The development of multiple *disjoint* representations of a common system under design (SUD) results in the model continuity problem [1]. Thus, the many different models representing the maturing design must be translated between tools. This translation increases the cost, the probability of errors, and thus design time.

Nowhere in the design cycle is the lack of model continuity more apparent than between performance and functional models. This discontinuity exists because the structure of the models for performance analysis and for functional design are very different [2]. Therefore, a separate performance model is maintained for the evolving design, rather than the design evolving from the performance model.

When designing a complex digital system, it is important to analyze the system's performance in the earliest possible stage of the design process. Typically, a performance model of the system is constructed and simulated for this purpose. A common approach is to utilize the performance model to aid in the selection of an architecture for the system. However, in today's design methods, the model is not used in later phases of the design process.

One possible solution to the model continuity problem is hybrid modeling. Hybrid modeling provides the capability of simulating abstract performance constructs and functional elements in a common simulation environment. Thus, hybrid modeling supports the stepwise refinement of abstract performance (uninterpreted) models into behavioral (interpreted) models. Once a part of the system is designed at the functional (or behavioral) level, it can be incorporated into the performance model, and the performance model is re-simulated in order to get a more accurate estimation of system performance. By repeating this process, element after element, a true stepwise refinement design process is achieved. Section 2 presents the hybrid modeling taxonomy that was jointly developed by Honeywell Technology Center and the Center for Semicustom Integrated Systems at the University of

Virginia. The taxonomy uses uninterpreted system and interpreted component attributes to categorize the construction of different hybrid elements. In particular, this section delineates those model attributes which fundamentally affect the development and implementation of the hybrid element interface. The linkage between model attributes and hybrid interface structure is also explained. The construction of different hybrid elements for different subclasses defined by the developed taxonomy is an ongoing research effort in both institutions. Well defined solutions and techniques have been developed and implemented for several subclasses and are presented in Section 3. Potential solutions for other subclasses that are under different stages of development are briefly discussed. Section 4 illustrates the hybrid modeling techniques via examples of system models. Finally, conclusions are summarized briefly in Section 5.

2. Hybrid Model Taxonomy

A *hybrid model* is a model that consists of both interpreted and uninterpreted elements. A hybrid model, therefore, provides a mechanism for the interchange of information between the two types of elements. This mechanism is called the *hybrid interface* and resolves the discrepancies between the domains of interpretation.

2.1 Hybrid Interface Structure

The function of the interface is to handle all interactions between the interpreted and uninterpreted elements, or in other words, to convert information from one domain to another. When information flows from an uninterpreted domain to an interpreted domain, the interface accepts tokens as input and generates data values (bits, integers, etc.) that match the data type of the input ports of the interpreted element. The interface has access to the color fields of the input token(s). The interpreted inputs can be considered as known values if they can be determined from the information carried with the token, or as random variables, depending on the level of abstraction of the hybrid model.

When information flows from an interpreted domain to an uninterpreted domain the interface accepts interpreted data values and releases tokens as outputs. In addition to data translation, this part of the interface has to provide the timing mechanism for releasing tokens to an uninterpreted domain. Typically, the number of interpreted element outputs will exceed the number of tokens that should be released. Thus, the interface must “bind” the interpreted element outputs to tokens in terms of value as well as timing information.

2.2 Hybrid Model Classes

This section defines an applicable taxonomy to adequately describe hybrid models and their interfaces. 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 development and implementation of the hybrid interface. The hybrid modeling space is partitioned according to the following characteristics:

1. The hybrid model *objective*
2. The *timing and synchronization mechanism* of the model
3. The *nature* of the interpreted element
4. The data transformation *mode*
5. The *data type* of the interpreted signals

Model Objective: Models are built with different objectives. These objectives strongly affect the structure and the functionality of the hybrid interface.For example, a model developed to examine only the temporal behaviors and constraints of the system will differ from a model designed to explore various functional implementations. The two major objectives are:

1. *Performance analysis* and *timing verification*:

Analyze the performance of the system when one or more components are modeled in the interpreted domain, and verify by simulation, that the system does not violate timing constraints.

2. *Functional verification*:

Verify by simulation that the function of the interpreted component is acceptable, within the context of the system model.

Performance analysis of a hybrid model, which results in more realistic performance estimation, is affected by the hybrid element in two different ways. The first one is due to the fact that the delay through the interpreted component itself is more realistic than the delay specified in the uninterpreted domain. This difference in delay affects the performance estimation of the system. The second way in which a hybrid element can affect system performance is due to some dependency of the rest of the uninterpreted model on the interpreted component output values. The interpreted component contains full functionality and, therefore, the values on its output signals may be used to alter the token flow through the model. Therefore, the performance analysis objective and the functional verification objective are related. Achieving both secondary objectives by using a single interface and a single technique is practical only if all input values to the interpreted component are known from the information within the tokens arriving from the uninterpreted model to the interface.

If the objective of the hybrid model is performance analysis only, the interface must detect when the interpreted element processing is completed, and release tokens to the rest of the model at the appropriate time. On the other hand, if the objective is functional verification, the interface operates on output values from the interpreted domain as well as timing. It has to be emphasized that this functional verification objective is only in the context of the performance model, which inherently does not include all system functionality.

Timing and Synchronization Mechanisms: The hybrid modeling technique depends upon the timing mechanism of the uninterpreted model. Thus, this attribute defines the timing and synchronization mechanism across the interface. The two types of system models are: *Synchronous* and *Asynchronous*.

Synchronous models usually refers to models of systems with a single global clock, i.e. the global clock synchronizes all operations within the system, and the model of such a system reflects this synchronization scheme. Asynchronous models usually refers to models of self-timed systems, in other words different parts of the systems are unclocked, or operate with different clocks or systems constructed of subsystems that communicate in an asynchronous fashion.

The functionality of the interface depends on the timing mechanism, especially in the case of multiple input token paths to the interface. Since the interface activates the interpreted element, multiple input token paths may be treated in several manners. For example, tokens may have to arrive at all input signals in order for activation or, the first token that arrives may activate the interpreted block. Hence, the interfacing technique is strongly influenced by the synchronization of the model.

Interpreted element: The hybrid modeling technique also strongly depends upon the type of the interpreted component that is introduced into the performance model. It is natural to partition interpreted hardware descriptions into combinational elements and sequential elements, however, research has suggested the following partition:

1. *Combinational Elements*:

Unclocked (with no states) elements, e.g. constructed of gates only.

2. *Sequential Control Elements* (SCE):

Clocked elements (with states) that are used for controlling data flow, e.g. a control unit or a controller

3. *Sequential Data-Flow Elements* (SDE):

Elements that include data-path elements **and** clocked elements that control the data flow, e.g. control unit and data-path.

For combinational interpreted elements, the outputs depend on the current inputs only and, therefore, the interface acts independently for each input token. On the other hand, for sequential interpreted elements, the interface must account for states, as well as inputs.

The major reason for partitioning the sequential elements into sequential data-flow and sequential control elements is based on the timing attributes of these elements. A sequential control element (SCE) is a cycle-based machine, i.e. control input values are read every cycle and control output values (that control a data-path) are generated every cycle. On the other hand, sequential data-flow elements (SDE) have data inputs and may have some control inputs but the output data is usually generated several cycles later. This difference in timing attribute will dictate a different technique for hybrid modeling.

Data Transformation Mode: The mode defines the data transformation/interface mechanism. When information is flowing from an uninterpreted domain to an interpreted domain, the values for the interpreted input signals can be generated in four different modes which are not mutually exclusive: Translate mode, Stochastic mode, External mode and State-based mode.

The translate mode simply converts data carried with tokens arriving to the interface. This mode is practical only if the tokens in the uninterpreted model contain sufficient information in order to generate interpreted inputs. The process of extracting data from color fields, interpreting them, and mapping onto the input signals of the functional model. Since the token must remain as small as possible for simulation efficiency reasons, the translate mode is somewhat restricted.

The stochastic mode allows the user to define and implement probabilistic generation of interface data. This mode operates in a manner similar to the performance modeling stochastic input stimulation. This capability is integrated such that ranges, distributions, etc. can be associated with other sources of data such as fields in the token or external information.

The external information mode supplies external data in the form of files to the interface. In this mode, data is extracted from files similar to how the translate mode extracts information from a performance token. The capability to filter, interpret, and manipulate external file data is integrated in this mode.

The state-based mode maintains internal state to handle complex interface interaction. In this mode, all the information to generate the functional or token-based input must be available at the interface. However, the state-based mode maintains internal state information, as well as a mechanism for traversing that state and generating appropriate interactions with the rest of the model.

Data Type: The data type defines the interpreted data generated or received in the hybrid interface. The data type depends on the interpreted element introduced to the hybrid model as well as on the modeling level of this element. It includes straight forward types such as bits, integer, reals, as well as more complex data types such as characters and enumerated data types.

2.3 Interfacing Scenarios

In addition to the attributes described above, the hybrid taxonomy consists of several possible interfacing scenarios. The interfacing scenarios define the data flow across domains of interpretation. To support both top-down and bottom-up design processes, four interfacing scenarios are possible and the potential usage of hybrid interface in these scenarios is described in Figure 1.

The scenarios are distinguishable by their environment. Several scenarios transfer data across a single boundary (U/I , I/U) and other move data from one domain and back (U/I/U , I/U/I). The U/I/U and I/U/I scenarios show insertion of a model of a different level into an existing model. In such a case, the inserted model is enclosed within an interface for both inputs and outputs. In the U/I and I/U scenarios, the overall model data flow is converted to another level.

During a typical top-down design process, the U/I/U scenario is very likely since an uninterpreted performance model is constructed followed by the insertion of various interpreted elements. This common scenario enables to preserve token information across the hybrid element. Whenever the interpreted block is surrounded by uninterpreted elements, it is regarded as a U/I/U interfacing scenario.

3. Hybrid Modeling Techniques

The hybrid taxonomy described above was jointly developed by Honeywell Technology Center (HTC) and the Center for Semicustom Integrated Systems (CSIS) at the University of Virginia (UVA), and it is common to both institutes with few minor exceptions. In general, the hybrid modeling techniques being developed are based on the same general methodology, but the actual implementations are different because of the differences in the performance modeling tools developed by both organizations.

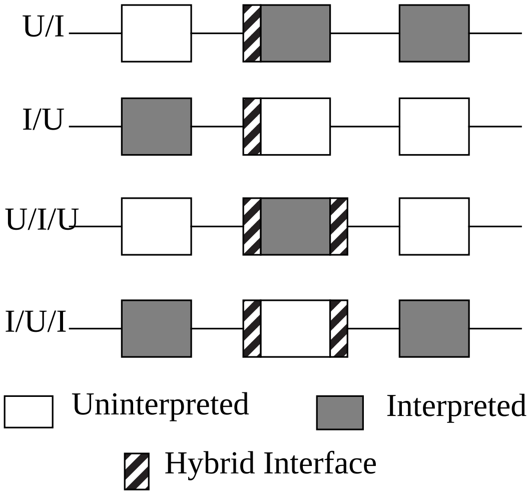

Figure 1 : Interfacing Scenarios

The differences between the Honeywell Performance Modeling Library (PML) and the UVa Advanced Design Environment Prototype Tool (ADEPT) can be traced to two different factors. First, the Honeywell PML is intended to develop performance models of systems that include more functional information than performance models developed in ADEPT. This inclusion of more functional information has the potential to ease the token to value translation process in hybrid modeling. This difference can be somewhat attributed to the different requirements to which the libraries were designed. The ADEPT library elements have a direct mapping to Petri Net components which allows more formal analysis techniques. The PML elements do not have that requirement.

Second, the actual implementation of tokens and token flow in the PML and ADEPT is different. For example, in PML, the tokens include routing information that is used to direct the flow of tokens over bus signals with multiple sources and/or sinks. In ADEPT, each signal has only one source and one sink, so routing information is not required. Some of the differences in implementation can be traced to the different levels of detail that are intended to be expressed in each tool as described above.

3.1 UVA techniques and future directions

Hybrid modeling is supported by the ADEPT environment, developed at the University of Virginia. In the ADEPT environment, a system model is constructed by interconnecting a collection of *ADEPT modules*. The modules model the information flow, both data and control, through a system. Each ADEPT module is implemented in VHDL and has a corresponding colored Petri Net representation, which is based on Jensen's CPN model [3]. The modules communicate by exchanging *tokens*, which represent the presence of information, using a uniform, well defined handshaking protocol [4]. Higher level modules can be constructed from the basic set of ADEPT modules. In addition, custom modules can be incorporated into a system model as long as the handshaking protocol is adhered to. The entire set of ADEPT modules is divided into six categories, of which the hybrid modules category is one of them. The hybrid modules support the development of the hybrid interface for hybrid models. A more detailed description of the entire ADEPT module set can be found in [5].

The hybrid modeling techniques implemented so far

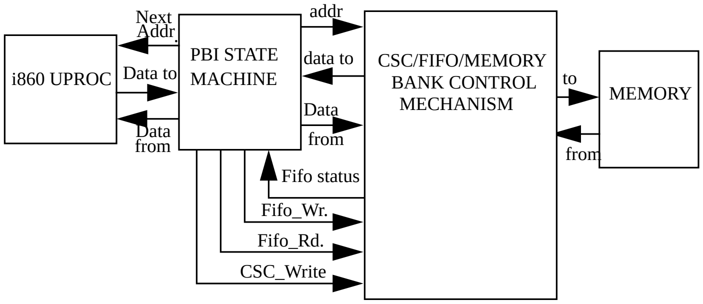

Figure 2 : Partitioned SMC Model

are concentrated on the performance analysis and timing verification objective. Therefore, a more realistic estimate of the system's performance is achieved by simulating the hybrid model as well as verifying timing constraints of interpreted elements. A comprehensive technique for models with combinational interpreted elements was developed and implemented within ADEPT. This technique includes the handling of the case of known inputs (translate and external modes of data transformation) as well as the case of unknown inputs [6][7]. For the latter, a statistic-based algorithms were developed and implemented to detect longest possible delays of combinational interpreted elements during hybrid model simulation.

The current effort in hybrid modeling is concentrated on models with sequential interpreted elements, given that the simulation objective is performance analysis. The technique for the case of known inputs is being developed. For sequential interpreted elements with unknown inputs, the technique is substantially different from that developed for combinational interpreted elements. This situation is the result of the fact that sequential elements maintain state, and system performance is determined by clock cycles as well as propagation delays. The research in hybrid modeling is mainly focused on this issue as well as developing techniques for the functional verification objective.

3.2 HTC techniques and future directions

Honeywell has developed several hybrid modeling mechanisms during the development of the PML. The processor model exhibits a form of synchronous interface with its software scheduler. The functional memory port of the processor model is a form of an external mode for hybrid interface mechanisms. The example described later in this paper is also a specialized hybrid interface. The focus of the Honeywell effort for RASSP hybrid modeling is to generalize these interfaces into a robust library. Honeywell intends to develop a set of building blocks which users can implement hybrid interfaces.

4. Examples

Typical examples of hybrid models and their usefulness are presented. The first one is from the UVA environment while the second is implemented in the Honeywell environment.

4.1 Hybrid Model of a Stream Memory Controller

A complex example employing these hybrid modeling techniques can be found in the design of a stream memory controller (SMC)[8]. The purpose of the SMC controller is to assist in obtaining peak memory bandwidth for vector processes by reordering the requested processor accesses in order to create a larger ratio of page hits when communicating with memory. A hardware implementation of a stream memory controller is under development at the University of Virginia. This particular implementation of the controller reorders the memory requests by organizing them in a FIFO buffer. This particular SMC is designed to interface to the Intel i860xp microprocessor. The SMC was selected as a hybrid example because the system is currently under development at UVA and information regarding design changes can be easily obtained and incorporated into the model.

The SMC was modeled using ADEPT at an uninterpreted level. The SMC design was partitioned into four sections, shown in Figure 2: the processor, the processor bus interface (PBI), the bank and fifo control

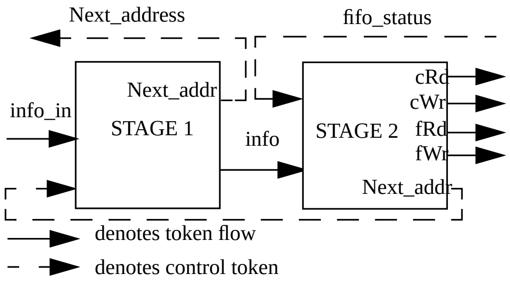

Figure 3 : Processor Bus Interface Model

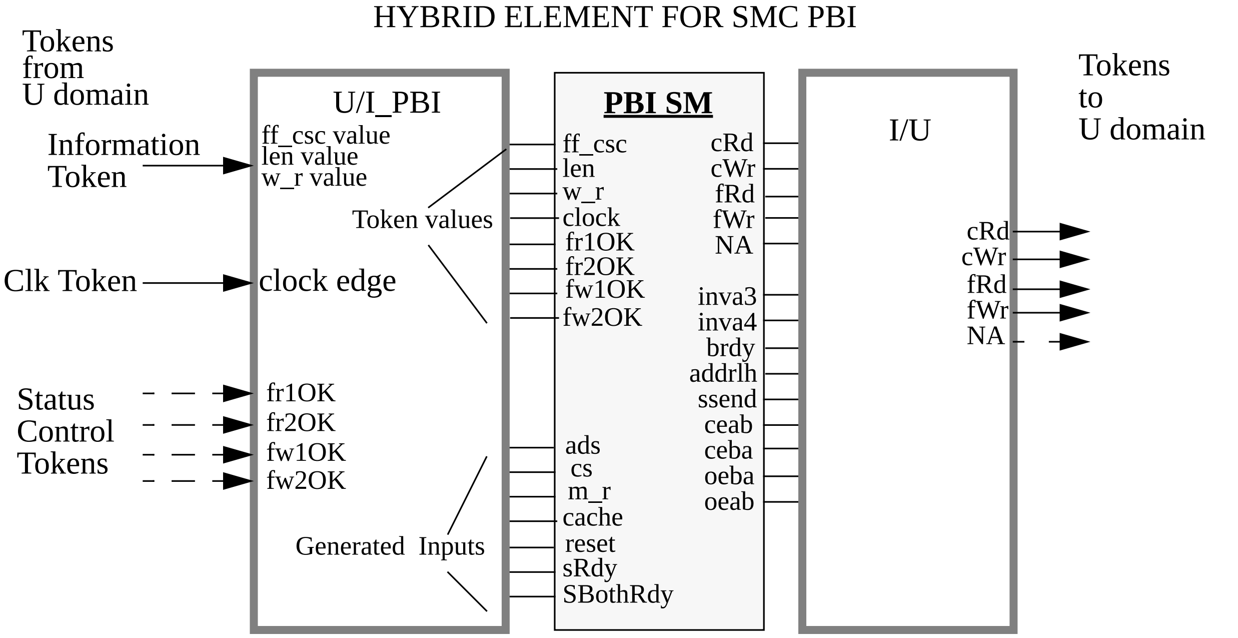

Figure 4 : Hybrid Element For SMC PBI State Machine

mechanism, and the page-mode memory. The PBI partition is of special interest for this model because it is the sole means of communication between the processor and the FIFO buffers. The PBI, which is shown in Figure 3, consists of a two-stage address pipeline network which supplies the FIFO buffer units with read and write information. The second stage of the PBI is a synchronous state-machine which is responsible for performing the read and write accesses to the FIFO buffers. This particular element has been designed using the Cascade[9] synthesis tool. As a result of this Cascade design, a VHDL behavioral representation of the state machine exists. This particular interpreted element was inserted into the uninterpreted model to form a hybrid model of the SMC.

This example demonstrates the hybrid modeling techniques for a synchronous system with sequential interpreted element. The objective of this model is performance analysis and timing verification. In this example, all interpreted inputs are known from the information carried with the tokens and the translate mode is the implemented mechanism for data transformation.

Figure 4 shows the hybrid element configuration for the PBI state machine. The hybrid element is a synchronous sequential hybrid element using known inputs.The U/I module maps the uninterpreted tokens to interpreted inputs and applies the explicitly determined values to the interpreted inputs. The interpreted component is the VHDL behavioral description of the PBI state machine. The I/U component maps the output values generated at the outputs of the interpreted component to tokens and releases these tokens back into the uninterpreted model.

The SMC hybrid model has been simulated to ensure proper operation and to study its performance in terms of achieving peak memory bandwidth. The results of this analysis can be found in Figure 5. The performance results showed that the hybrid model of the system found a slightly lower percentage of peak memory bandwidth for the SMC. This result is due to the fact that the behavioral description of the PBI state machine is more accurate than in the uninterpreted model. The lower performance metrics are resultant of a more detailed handshaking procedure between the PBI state machine and the FIFO buffers.This refinement of the synchronous interface for the PBI state machine resulted in a slower, but more realistic model of the design.

In this particular example, the advantage of using hybrid modeling techniques to help guide model refinement is demonstrated through the hybrid model's

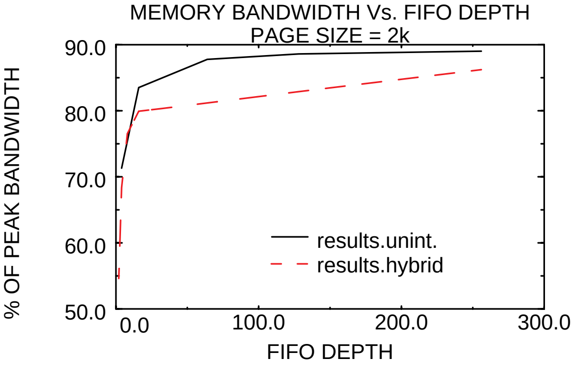

Figure 5 : SMC Performance Results

improved modeling of the PBI interface synchronization. By having the ability to guide model refinement through hybrid modeling techniques, design inconsistencies and performance bottlenecks can be identified early in the design process.

4.2 Hybrid Model of a 3-D Rendering Pipeline

Honeywell is actively involved in the definition of the next generation display processors for military and commercial cockpits. One such architecture, the Cockpit Display Generator (CDG) [13][14], provides the graphical and video processing power needed to drive future high resolution display devices and generate more natural panoramic 3-D formats. The CDG will provide multichannel, high performance 2-D and 3-D graphics, and real-time video manipulation. The area of greatest challenge in this design was the 3-D rendering pipeline. A modified version of the Pixel-Planes [10][11] was chosen for the pipeline. This SIMD approach gives each pixel memory its own computational hardware. A key element in this architecture is the Enhanced Memory Chip (EMC) [12]. The EMC is essentially a memory that is enhanced with tightly coupled computational logic on the same chip. The logic is arranged in a SIMD architecture that can efficiently perform the linear expression evaluation that is common in the lowest level graphic rasterization.

The performance/functional hybrid model of the CDG 3-D rendering pipeline is shown in Figure 6. Several processors in the 3-D pipeline are modeled using the performance processor model from the Honeywell PML. This model, described in another paper in this proceedings [15], provides the capability of hosting functional tasks on an abstract characterized model of a processor. Detailed algorithms of the various steps in the 3-D pipeline were described and modeled. Different processor characterizations were used to determine the appropriate processor for each step in the pipeline. Initially the EMC function was modeled in this manner. However since this is a key function of the pipeline, and requires a custom solution, a more detailed simulation was required to verify the configuration and functionality required. The capability to perform this analysis is central to a hardware/ software co-design activity.

The Solid State Electronics Directorate at Wright Labs designed and fabricated an EMC [12] using a 0.35 micron CMOS process. The RTL/gate level model of this chip provided a detailed view of this key component. An array of EMCs are required for a complete graphics system. Honeywell performed simulations on various architectures to determine the appropriate EMC configurations and control mechanisms. These simulations will help determine the requirements by ELED for the next version of the EMC which can then be used in the CDG implementation. These simulation experiments on the functional EMC will help decide the number of on-board ALU’s (64, 32, or 16) for the chip. Two primary characteristics are being examined: 1) test the number of ALU’s that are enabled per triangle, and 2) test the number of bits of on-board memory that are used per triangle. Two test cases for exercising these models have been developed: a data set that has a very consistent set of triangles sizes (top down God’s Eye view) and a data set that has a range of triangle sizes (low grazing angle). These two test cases establish a mix of triangles that represent the two extremes associated with rendering terrain data perspective views. These simulations are being done using a functional model of the EMC which was developed by Honeywell and verified against the RTL model. This model allowed rapid simulations of the 3-D pipeline/EMC hybrid model.

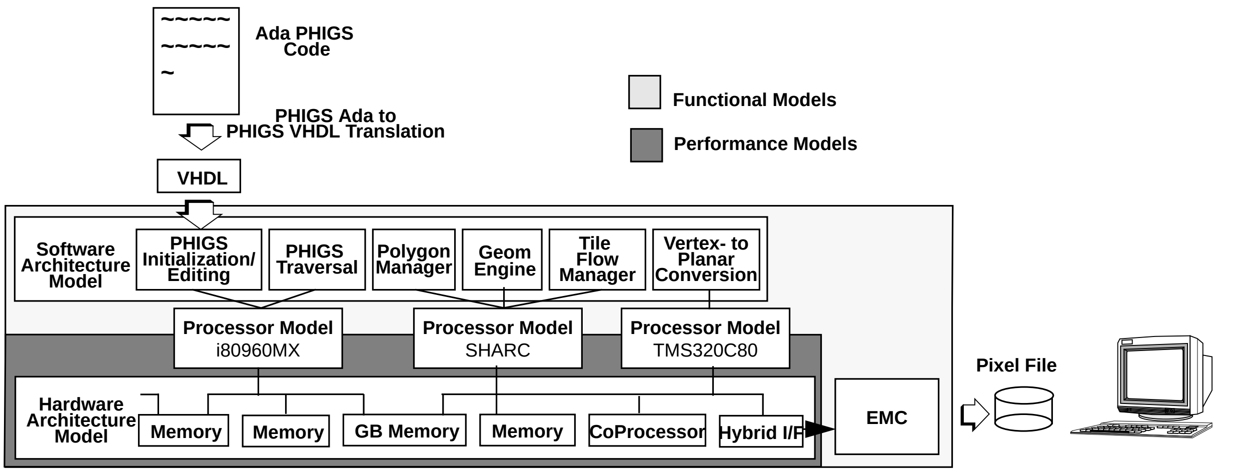

Figure 6 : 3D Renderer integration with gate-level EMC

This functional/performance model example highlights two hybrid modeling cases; the software scheduler-processor model interface and the functional memory port. Both of these cases allow detailed functional models to interface with a performance model of a processor. This allows rapid, detailed analysis of a portion of the design while keeping a known element, the processor, at an abstract level. Specifically in the CDG/EMC case, the hybrid model allowed rapid verification of detailed chip requirements at an early point in the design process.

5. Conclusions

Hybrid modeling provides the capability of simulating several levels of abstraction and interpretation in a single model; starting from the uninterpreted level used for performance analysis down to behavioral or functional levels (interpreted) used in the design and implementation process. Thus, hybrid modeling supports a top-down design flow by providing the capability for stepwise refinement of performance models into behavioral models. Its advantage is in verifying design decisions at an early stage of the design process, and it enables the designers to asses the impact of different component implementations. By having this ability, potential performance bottlenecks can be identified early in the design.

Hybrid modeling remains an ongoing research area in both the Center for Semicustom Integrated Systems, at the University of Virginia and Honeywell Technology Center. This paper has presented a framework, along with the current implementation of the modeling environment, for the development of hybrid models. These results represent a significant advancement to the state of the art in hybrid modeling technology.

6. References

[1] Franke, D., Purvis, M., “Hardware/Software Codesign: A Perspective”, *13th International Conference on Software Engineering*, May 1991, pp. 344-352.

[2] Jain, P., “Myth and Realities of System Design”, ASIC & EDA, Jan 1993, pp. 17.

[3] K. Jensen, “Colored Petri Nets: A high level language for system design and analysis,” in *High-level Petri Nets: Theory and application*, K. Jensen and G. Rozenberg (Eds.), Berlin: Springer-Verlag, 1991, pp. 44-119.

[4] Aylor, J., et al., Performance and Fault Modeling with VHDL, J. Schoen Editor, Prentice-Hall Inc., pp. 22-144, 1992.

[5] Cutright, E., et. al. “A Handbook on the Unified Modeling Methodology Building Block Set”, 2nd Edition, University of Virginia, Technical Report No. 910828.1, May 1993.

[6] MacDonald, R., “Hybrid Modeling of Systems with Interpreted Combinational Elements”, Ph.D. Dissertation, University of Virginia, May 1995.

[7] MacDonald, R., Williams. R., Aylor, J., “An Approach to Unified Performance and Functional of Complex Systems”, IASTED Conference on Modeling and Simulation, April 1995.

[8] W. Wulf, C. Hitchock, S. Moyer, S. McKee, “Increasing Memory Bandwidth for Vector Computations”, University of Virginia, Computer Science Report. November 1992.

[9] EPOCH, User and Reference Manual, Cascade Design Automation, Inc., 1993.

[10] Fuchs, H., Poulton, J., et al., “Pixel Plane 5: A Heterogeneous Multiprocessor Graphics System using Processor Enhanced Memories,’’Computer Graphics Proceedings, Vol.23, No.3, July 1989, pp.79-88.

[11] Fuchs, H.,J. Poulton, A.State, “An Architecture for Advanced Avionics Displays,’’ University of North Carolina at Chapel Hill, WRDC-TR-90-7006 (Technical Report to Wright Laboratory), May 1990.

[12] Bilik, S., “The Modeling and Simulation of a SIMD Graphics Engine in VHDL’’, Masters thesis, Wright State University, 1993.

[13] Hancock, W., M. Johnson, J. Rogers, J. Ghrayeb, “Meeting the Graphical Needs of the Electronic Battlefield,’’ DASC Prodeedings, Dayton, OH, May,1994.

[14] Johnson, M., J. Rogers, W. Hancock, D.Iacovou, F. Rose, H.Spaanenburg, P.Dietrich, J. Ghrayeb, “A Single Card Real-Time 3-D Rendering Engine,’’SPIE Proceedings, Orlando FL, April, 1994.

[15] Carpenter, T., F.Rose, J.Shackleton, T.Steeves, O.von der Hoff, “Evaluating Distributed Multiprocessor Designs,’’ RASSP Conference Proceedings, Washington, D.C., July, 1995.

7. Acknowledgment

The Honeywell work has been sponsored by the following contracts, F33615-94-C-1495 (VHDL Hybrid Models) and F33615-92-C-3802 (CDG). The authors wish to thank Todd Carpenter, Paul Dietrich, Jeff Groat, Henk Spaanenburg, and Todd Steeves, who comprise the CDG team, for their support and contributions to the hybrid modeling portion of the CDG design. We wish to acknowledge particularly the design talents of Todd Carpenter and Todd Steeves which made the PML possible. We also wish to thank Dr. John Hines and his staff, and Capt.Ken Runyon from Wright Labs for their support.

The UVA work has been sponsored by contract F33615-93-C-1313 (ARPA) and partially by contract B09332526 (Honeywell).The authors would like to acknowledge all those who have contributed to the development of the ADEPT environment, in particular, Ramesh Rao, Sanjay Srinivasan, Eric Cutright, Maximo Salinas, Gnanasekaran Swaminathan, Rich MacDonald and Sanjaya Kumar.