AN APPROACH TO HARDWARE/SOFTWARE CO-MODELING FOR RAPID DESIGN EXPLORATION

Attention: Please Read This Page

This paper was originally presented at the **High-Level Electronic System Design Conference** held from October 7-9, 1997. While we believe that the material is important and will be interesting to current users of Foresight Systems software, we have been through some changes since the paper was prepared.

Nu Thena Systems, Inc. was a predecessor company to Foresight Systems M&S. All references to Nu Thena are obsolete. Readers should assume that Foresight Systems M&S replaces all instances of Nu Thena Systems, Inc.

The current contact information for Foresight Systems M&S is:

Foresight Systems M&S
12550 N. 89th St.
Scottsdale, AZ 85260

http://www.foresightsystems-mands.com
fs_marketing@foresightsystems-mands.com

The current production versions of the software do not support the direct synthesis to conventional hardware description languages. **FS/CoderC**++ does automatically translate graphical Foresight executable system models into C++ code, which can be compiled and linked with a run-time library to form stand alone executables.

The diagrams and illustrations included here were prepared with versions of the software current at the time that the paper was prepared. Some of them may not be identical to equivalent images from the current version of the software. We apologize for any potential confusion.

Please excuse the use of any out of date expressions and terminology. The authors believed that their application of technical terms was consistent with standard practice at the time that this document was written.

If you have questions, concerns, or issues, please contact:

Dean Stevens
President, Commercial Systems Group
Foresight Systems M&S
deans@foresightsystems-mands.com
(408) 278-3983

HESDC '97 Proceedings

High-Level Electronic System Design
Conference
October 7-9, 1997
San Jose, CA

Organized by
MultiDynamics, Inc.

AN APPROACH TO HARDWARE/SOFTWARE CO-MODELING FOR RAPID DESIGN EXPLORATION

Mike Vertal and John Crowley
Nu Thena Systems, Inc.
1430 Spring Hill Rd., Suite 220
McLean, VA 22102 USA
(703) 356-5056 /0498 (fax)
mikev/johnc@nuthena.com

ABSTRACT

Electronic system designers must rapidly explore numerous design alternatives in order to satisfy stringent product requirements while fully exploiting available implementation technologies. This paper describes an approach for modeling integrated hardware/software systems which supports rapid design space exploration. Our method is based upon a set of flexible and expressive architectural modeling constructs, which enable efficient model development and trade-off analysis. Our approach has been implemented within a commercially available toolset, and successfully employed on numerous embedded system applications.

INTRODUCTION

The proliferation of electronic systems and their expanding utility in al1 types of everyday activities continues unabated. The large breadth of potential applications enabled by advances in implementation technologies, such as systems-on-a-chip silicon integration, provides today's developers with numerous opportunities to capture a lucrative share of new markets. Dominant market share, however, usually goes to those first to market with the right products. Therefore, system-level design activities are becoming a key part of development processes, since decisions made at the early stages of development have the largest impact upon schedule and product cost, functionality and performance. More specifically, given the increasing proportion of embedded software in electronic system implementations, hardware/software co-design (Wolf 1994) is gaining acceptance as a key design activity that can significantly raise design productivity and enable full exploitation of today's rapidly advancing implementation technologies.

Hardware/software co-design consists of three primary activities:

- Functional verification
- Architectural design
- Component synthesis and co-verification

Functional verification addresses the need to verify system requirements. Capturing and analyzing a system design based upon customer requirements, independent of any hardware and/or software implementation details, allows designers to verify that the functionality of the system satisfies market demands. Functional verification is an important step toward getting the "right product" developed.

Architectural design involves the development of an integrated collection of (hardware, software, interface) components that will implement the system functionality. A key activity of architectural design is exploration of the hardware/software design space, which entails the analysis of numerous potential architectural implementations in search of a suitable design. In particular, an architecture must be realized which implements the required functional behavior while satisfying the nonfunctional requirements such as system performance, cost, power, and size.

Once an architecture is designed, synthesis of its constituent components into detailed hardware and software designs can proceed. The component designs must satisfy the architectural design requirements and must be implemented in available technologies. Early verification of component designs can be accomplished with hardware/software co-verification, which is supported by tools such as Seamless CVE™ (Bailey and Leef 1996) that allow designers to verify software component design implementations before physical electronic hardware is available.

Verifying system functionality, designing an integrated architecture, and synthesizing and verifying the detailed components represents the essential activities of hardware/software co-design. Performing these co-design activities in the presence of time-to-market pressures requires that they are executed rapidly and efficiently. Consequently, today's embedded system designers need robust tools to facilitate these key co-design activities.

This paper describes an approach for modeling integrated hardware/software systems which enables rapid design space exploration. Our particular area of focus is interactive, simulation-based design exploration which enables trade-offs related to system performance. Other complementary methods of design exploration, such as analytical methods used to evaluate power and packaging trade-offs (Stoaks 1997) may be used in conjunction with our approach. We have implemented our techniques in a commercially available toolset, Foresight™, which supports a simulation-based approach to system level design (Vertal 1994). The following section provides a survey of the history of electronics systems design, and motivates the need for a new approach to modeling today's integrated hardware/software systems. Subsequent sections cover the details of our modeling approach, and an overview of the implementation of our techniques in the Foresight toolset.

EVOLUTION OF ELECTRONIC SYSTEM DESIGN

Over the last decade, electronics-based system design has evolved from bottom-up to top-down design methods. Early on, electronic system design consisted of first specifying the system requirements, and then designing an implementation from the bottom-up using discrete off the shelf components. Digital simulation or breadboard prototyping proved sufficient for verifying the correctness of the design with respect to the requirements. With the introduction of programmable logic, FPGAs, and ASICs — along with associated hardware description languages — a top down design approach proved advantageous by allowing the designer to describe and verify functionality first with behavioral, simulation, and then automatically synthesize an implementation onto the target technology. In general, top down design methodologies typically verify functionality first independent of architectural detail. Functionality is then incrementally refined and architectural detail is incrementally added, until a final architectural implementation is realized.

With the recent advent of highly complex, tightly integrated hardware/software-based systems, a new design paradigm is needed to address the particular design challenges of these systems. The growing interest in this area is evident by various highly publicized efforts. Notable efforts include the Virtual Socket Interface (VSI) alliance and the Systems Level Design Language (SLDL) standardization efforts. VSI (Waller 1997) is aimed at defining a common interface standard for reuse-based design utilizing various sources of intellectual property (IP). SLDL (Goering 1997) is a complementary effort aimed at defining a standard language suitable for embedded systems which is higher in abstraction than current hardware description languages. Our modeling approach is compatible with these efforts, in that our approach supports IP-based design reuse and is independent of the syntax of any particular SLDL. Our approach may also contribute toward defining semantic requirements for an SLDL's architectural modeling capabilities.

We advocate a design process which is based upon the successful top down design methods - functional verification via executable modeling, design abstraction, and design reuse - but is enhanced to address the complexities of embedded systems. Specifically, our approach separates a single top down view into two separate but related, orthogonal views: one for functionality and one for architecture, each of which are executable, support multiple levels of design abstraction, and enable IP-based design reuse.

A HARDWARE/SOFTWARE CO-MODELING APPROACH

Our approach splits the typical top down design model into two orthogonal top down design models — one describing the functional requirements independent of hardware or software details, and another describing architectural details of a particular hardware/software implementation. Each hierarchical model describes a separate view of the system to support clear reasoning about each independently. One of the main benefits of this separation is the support of rapid design exploration. Our approach is similar to the modeling methodologies described by others (Hatley and Pirbhai 1987; Ward and Mellor 1987; Gajski et al. 1994), but extends them to enable executable architectural modeling, which in turn facilitates simulation-based design space exploration.
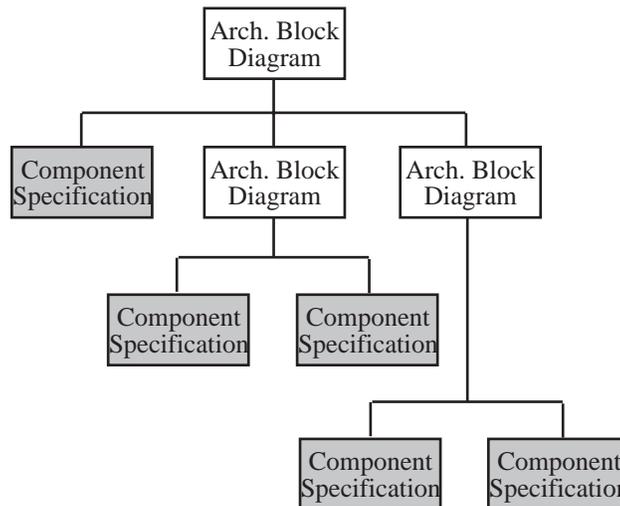
Functional modeling

The functional model captures system behavioral requirements in an executable form, suitable for verification of their correctness. This model is mainly independent of hardware or software implementation details, enabling the system designer to initially concentrate on verifying the correctness of functional requirements. Its hierarchical nature (shown in Figure 1) supports description at multiple levels of detail, and also enables IP blocks to be inserted at any level of the hierarchy. At the lowest levels of the functional hierarchy reside primitive functional objects which describe detailed behavior. To facilitate design reuse and rapid model development, any or all of these functional blocks may be parameterized reusable elements.
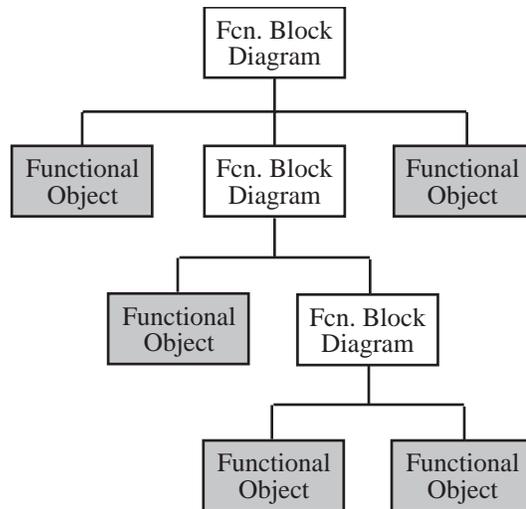
Figure 1. A Functional Model Hierarchy

<u>**Architectural modeling**</u>

The architectural model describes the collection of components - hardware components, software processing components, and interface components - which will implement the system's functional behavior. All hardware and software implementation details are contained within this view. This hierarchical model incorporates models of architectural components that can be described at multiple levels of detail, thus enabling a variety of design analysis ranging from high level architectural performance to more detailed analysis of components and protocols. A high-level abstract architecture model consists of components which are characterized by resources and queues, and could be used to identify bottlenecks. This coarse level of architectural analysis can then be refined to support more detailed design exploration and analysis. At the lowest levels of the architectural hierarchy reside primitive blocks which describe component behavior, as illustrated in Figure 2. And similar to the functional model, any or all of these architectural blocks may be parameterized reusable elements to facilitate design reuse and rapid model development.


Figure 2. An Architectural Model Hierarchy

<u>**System Partitioning**</u>

To complete the system design model, the two independent views of the system are linked by a partitioning specification that describes the mapping of functional objects onto architectural components, as shown in Figure 3. The combination of a functional model, an architectural model, and a mapping between them comprises a single system design alternative. Taking the

view that the functional model has been verified and correctly represents the system's behavioral requirements (and therefore is fixed), the design space is defined to be the union of the set of all possible architectural configurations and the set of all possible mappings of functions onto architectural components within a given architectural description.
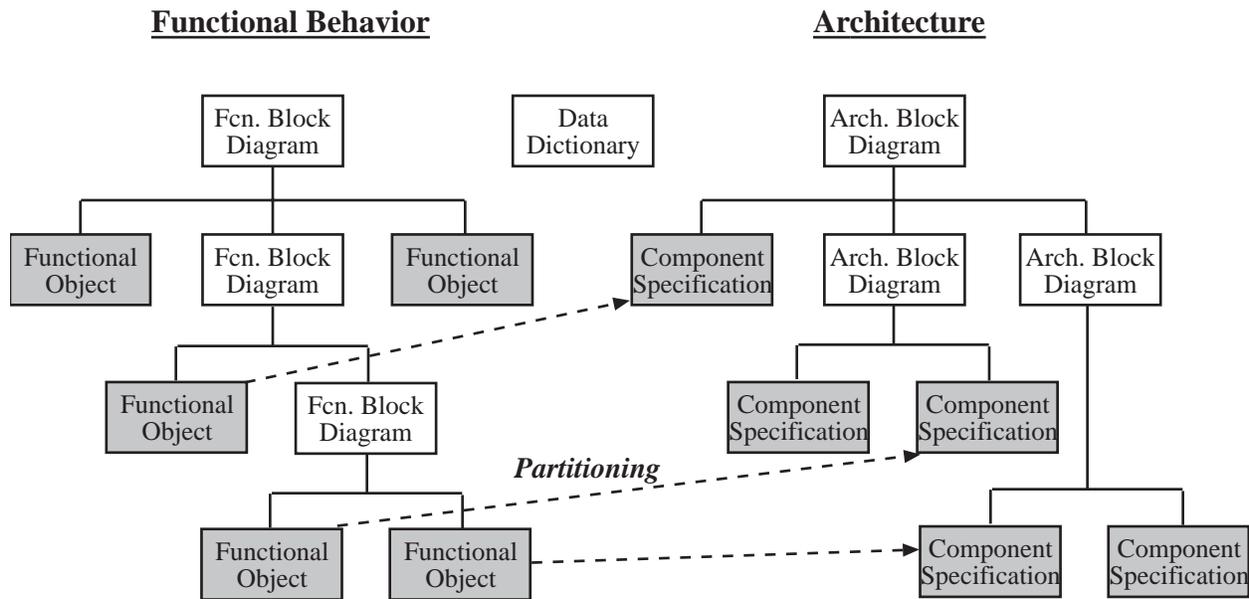
Figure 3. Functional Partitioning Completes the System Design Model

Design Space Exploration

Design exploration is the navigation of this design space in search of a suitable architecture that implements the required system functionality within the limits imposed by the design constraints. The time it takes to explore design alternatives is primarily composed of engineering time spent creating and modifying the architecture model, partitioning the functional objects onto the architectural components, simulating a given system design alternative, and analyzing simulation results. Any attempt to accelerate the exploration process must address at least one of these areas.

Our approach targets each of these to different degrees, with most emphasis on reducing the time for both the modeling and simulation activities. Specifically, we reduce modeling time by supporting an efficient modeling approach that—due to the separation of concerns of function and architecture, and with parameterized reusable libraries—allows the designer to easily focus in on a particular area of interest. Simulation time is reduced by supporting multiple levels of design abstraction, on both the functional model as well as the architectural model. By incorporating design detail only where necessary, simulation run time can be minimized. For example, low risk areas of the system design can be described abstractly thereby permitting fast simulation. Support for parameterized libraries also facilitates rapid model construction and modification. By encouraging design reuse, models may be quickly constructed by choosing from libraries, and their behaviors may be quickly modified between simulation runs by changing parameter values.

Most conducive to rapid design space exploration, though, is the orthogonality of architecture and functionality. By separating functions from implementation details, architectural components (processors, buses, memories, etc.) can be swapped in and out of the architecture model without any impact upon the structure of the functional behavior model. Similarly, functional partitions can be quickly specified and modified without any impact upon either the functional model or architecture model. The foundation of our co-modeling approach, that which is required to achieve independent but related functional and architectural views, is the executable architectural model. In particular, the components within the architectural model must be 1) independent of, but driven by, functional objects, 2) interconnected with flows that characterize architectural dependencies such that implementation details can be hidden from the functional model, and 3) able to be described at various levels of detail. How we have achieved this type of architectural modeling, and a resultant rapid design exploration process, is described in the following section.

DESIGN EXPLORATION WITH FORESIGHT

Our modeling approach to systems design is supported by the Foresight toolset, which incorporates a systems-level design language that enables: 1) functional modeling and requirements verification, 2) architectural performance modeling and analysis,

3) partitioning of functional objects onto an architectural design, and 4) parameterized reusable elements for both functional and architectural modeling.

Functional modeling with Foresight

Foresight functional models are built hierarchically from a collection of modeling constructs. Foresight's primary modeling construct is a hierarchical block diagram, an example of which is shown in Figure 4. Foresight's executable block diagrams have their origins in the real-time extensions to structured analysis (Hatley and Pirbhai 1987; Ward and Mellor 1987), with the important distinction of support for model executability. These block diagrams allow systems to be described as a network of communicating processes, which are well suited for modeling embedded systems given their high levels of functional complexity and their inherent concurrency.
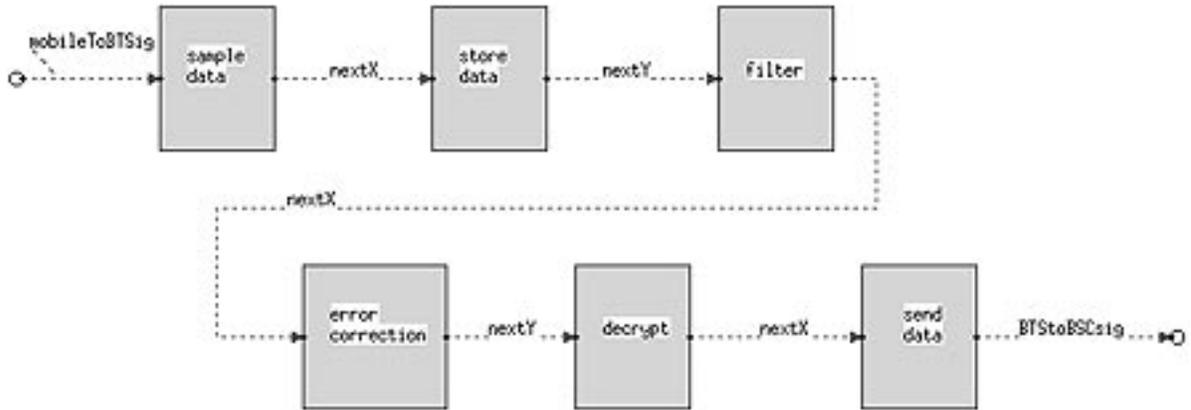
Figure 4. A Foresight Functional Block Diagram

Process blocks within a Foresight block diagram may be decomposed hierarchically into more block diagrams, or with Foresight primitive, executable modules: state machines, mini-specs, pre-defined library elements, or user-defined reusable elements. These primitive elements have the expressive power to capture essentially any type of functional behavior, at any level of detail. For example, mini-specs are described with a full procedural modeling language, which incorporates a complete VHDL-like procedural language embellished with numerous built-in function and procedure calls for systems modeling. State machines may be arbitrarily complex, and may include full mini-spec behaviors on their state transitions. All process blocks are connected with control and/or data flows, whose data types are stored and maintained in a data dictionary that supports a full range of types, including bits, integers, reals, enumerations, strings, records, and arrays.

By default, functional objects take zero time to compute and data is transferred between objects with zero delay, which allows for pure functional models to be built. These functional models, which may incorporate timing information to capture behavioral requirements, are used for requirements verification. Most of a system's timing behavior (e.g., computational delays, transport delays) is typically introduced after requirements verification. Timing behavior due to architectural component constraints is incorporated by constructing an architectural model and partitioning functional objects onto the architectural components.

Architectural modeling with Foresight

Foresight supports high level architectural modeling with numerous built-in modeling constructs and libraries tailored for architectural performance analysis. Similar to other dedicated performance modeling tools and simulation languages, Foresight incorporates constructs such as queues (with various scheduling mechanisms), resources, and random number generators to model high level architectural behavior.

Foresight primitive resource library elements comprise the basis of its architecture modeling capabilities. The two primary types of resource elements are:

1) Quantity-based Resource—characterized by a capacity of some resource, used to model components which have finite quantities of elements, like memories;

2) Processing-based Resource—characterized by a work rate and a scheduling algorithm, used to model components that process or transfer information, like CPUs and buses.

These parameterized resource elements may be used to model architectural components. For each, the capacity (or work rate) can be finite, and therefore contention for the resources and the effects of resource bottlenecks can be identified. Alternatively, the capacity (or work rate) can be infinite, which is useful to identify the "ideal" or maximum amount of architectural resources that are needed.

To support design exploration, functional objects (within the functional behavior model) may be partitioned onto one or more of these resource elements which characterize an architectural component. During model simulation, a partitioned functional object that is due to execute will in turn execute its allocated resource(s), the high-level performance behavior of which will determine the amount of time that the functional object will take to execute. Consider the example of a functional object representing a software thread (we'll call it Thread_A) mapped to a CPU (processing-based) resource with a pre-emptive priority based scheduling algorithm. When Thread_A is activated within the functional model, it will in effect trigger its CPU, at which point the CPU will attempt to execute the thread. If at that instant there is no other thread running on the CPU, Thread_A is executed immediately. As long as no other higher priority thread requests the CPU during Thread_A's execution, then at the end of its execution time and Thread_A's amount of work, the functional object will complete its function and generate its outputs at the end of the elapsed compute time. At a minimum, this compute time is based upon the specified CPU work rate and the estimated amount of work required by the functional object, so that with a CPU work rate of 10 MIPS and a Thread_A estimated amount of work of 30 thousand instructions, the resultant compute time would be 3.0 milliseconds (ignoring for this example any context switch times). Of course, the more interesting case is where there are multiple threads (with different priorities) competing simultaneously for the same resource, or when multiple resources are required to implement a functional object.

While these resource elements (along with other performance modeling elements like queues) comprise the most elementary part of architecture modeling, they do not entirely satisfy our need for an architecture model; that is, an executable architecture model consisting of interconnected components, which are independent of, but driven by, the functional model. For example, consider the case when multiple resources are required to implement a functional object, such as a software thread needing access to a CPU as well as memory (which may in turn may imply access to a shared bus). In this case, the relationship between the functional model and the architectural model of individual resources goes beyond that of a straightforward mapping, requiring the designer to incorporate architectural details within the functional model. This intermixture of the system functionality with implementation issues negatively impacts design exploration efficiency. Also, while using pre-defined resources to model components is suitable for high-level trade-off analysis, some cases require more detailed performance analysis. Examples include evaluating different protocols on a communications channel, or analyzing various cache strategies for a CPU. To surpass these limitations and support completely independent function and architecture modeling, along with detailed modeling of component behaviors, we introduce a more powerful architectural modeling mechanism—user defined resources.
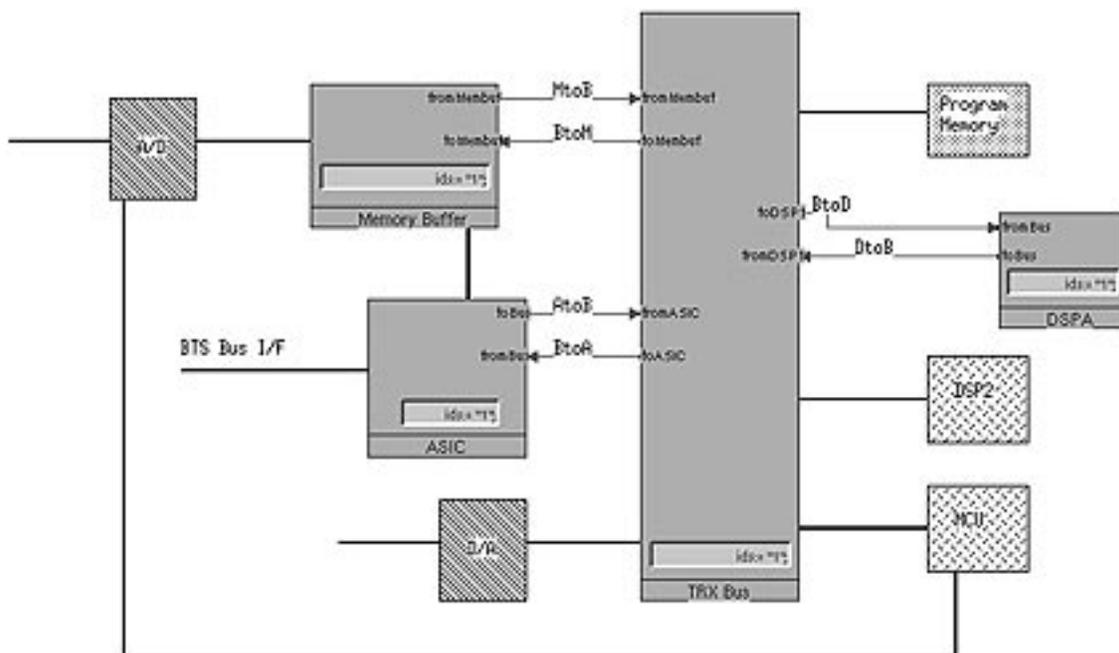

Figure 5. A Foresight Architectural Performance Model

User defined resources (UDRs) retain the essential characteristic of the elementary, pre-defined resource elements in that their execution can be driven by a partitioned functional object. Unlike a pre-defined resource, though, their behavior is not predetermined but rather can be defined with Foresight's full functional and resource modeling constructs. As a result, arbitrarily complex architectural component models may be built. Furthermore, UDRs may be interconnected to form a complete, executable architecture model. And because of their ability for interconnection, UDRs may trigger additional UDRs to execute as dictated by the architectural component's behavior, allowing the designer to separate the architectural implementation details from the functional model. For example, a DSP processor needing access to a bus and a bank of memory to access data will do so entirely within the architectural model, without any impact upon the functional object's description—in effect, hiding all implementation details from the functional specification. Therefore, partitioning of functional objects may be specified rapidly, functional objects themselves may be mixed and matched, and architectural elements may be easily substituted without impacting the functional specification. Figure 5 illustrates an example of an executable architectural model comprised of interconnected Foresight UDRs. This model (of a cellular base transceiver unit) consists of a Memory Buffer, ASIC, and DSP connected via a Bus (ignoring the shaded boxes which are non-executable).

UDRs substantially extend Foresight's elementary resource and queueing library elements with the ability to define architectural models at varying levels of abstraction, and to enable the creation of individual component models that may be interconnected to separate the implementation details from the functional model. Foresight UDR elements allow us to implement an architectural modeling approach suitable for rapid design exploration. They enable the construction of architectural models which are:

- executable,
- comprised of interconnected components,
- independent of, but driven by, a functional model, and
- accommodate reuse-based design.

Designing with Foresight

As shown in Figure 6, the two orthogonal views of the system—function and architecture—are linked by a partitioning specification that describes the mapping of functional objects onto architectural components. With Foresight, functional objects are partitioned onto resource specifications via function calls intrinsic to the mini-spec language.
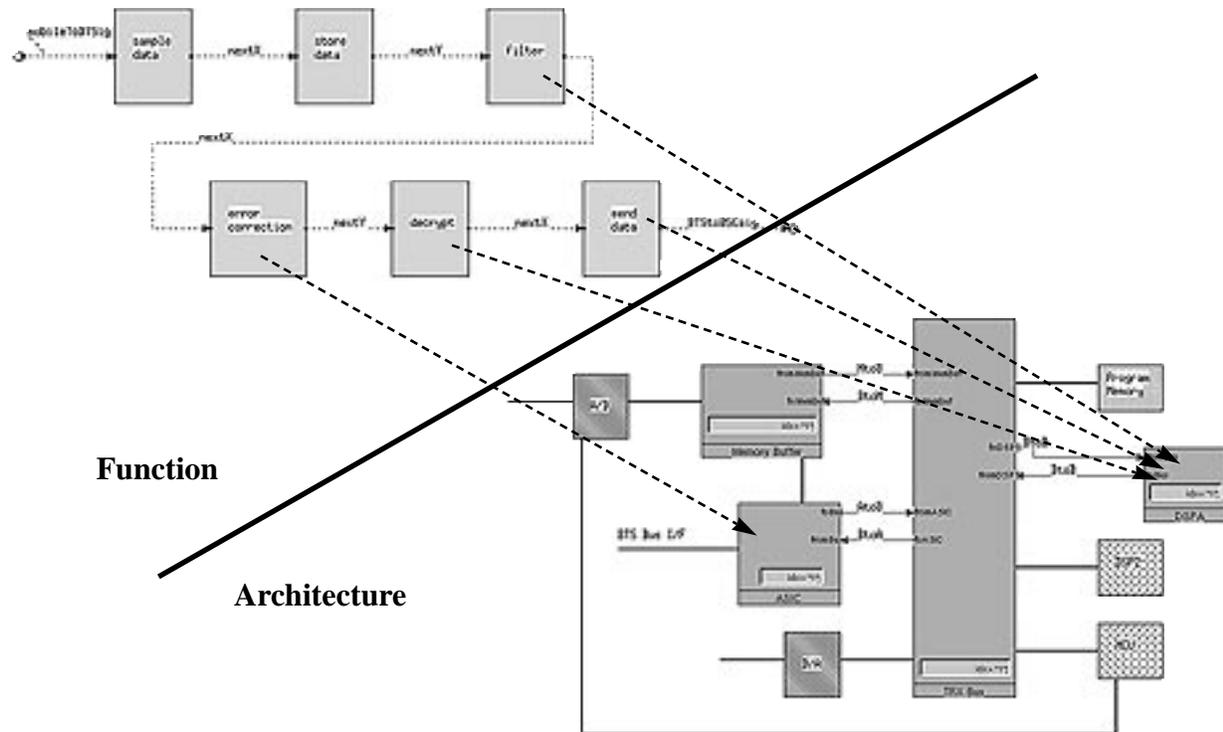

Figure 6. Foresight System Design - Hardware / Software Trade-offs

The combination of a functional model, an architectural model, and a mapping between them comprises a single system design

alternative. Figure 6 illustrates a functional partition of a cellular phone base station's signal processing functions (recall Figure 4) onto the transceiver unit architectural model. A simulation of a system design alternative enables a designer to determine whether performance constraints are satisfied. If constraints are not met and a design change is warranted, repartitioning functionality (e.g., moving the decrypt function from software (i.e., DSP) to hardware (i.e., ASIC)) can be easily achieved by simply changing a single parameter in a function call. Architectural details associated with the change are hidden within the architecture model. When this example is extended to a larger scale design, where numerous parameters specify the functional partitioning, these parameters can be stored externally in files. Separate files containing different partitioning specifications may then be used as input to batch simulations to explore a design space rapidly.

SUMMARY

An approach for modeling the functional behavior and architectural design of integrated hardware/software systems was presented. A key benefit of our orthogonal functional/architectural modeling approach is support for rapid evaluation of numerous design alternatives. The concept of user-defined resource specifications—which are independent of, but driven by, functional objects—enables the creation of executable architectural models that allow the system designer to hide implementation concerns from functionality. This separation of concerns provides a framework for rapidly

- configuring architecture models,
- partitioning functional objects onto architectural components,
- inserting component models at various levels of detail, and
- utilizing reusable elements.

All of which contribute toward rapid model development, accelerated simulation execution, and most importantly rapid design space exploration of integrated hardware/software systems.

REFERENCES

Bailey, B. and S. Leef "Making the Shift Toward Integrated System Design" *Electronic Design,* July 8,1996.

Gajski, B., F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems,* pp. 86-88 and pp. 171-200, P T R Prentice Hall, Englewood Cliffs, New Jersey, 1994.

Goering, R., "Design Language Will Speak Many Tongues" *EE Times,* pp. 1,14, July 21 1997.

Hatley, D. and I. Pirbhai, *Strategies for Real-Time System Specification,* Dorset House, New York, 1987.

Solanti, P. and M. Vertal, "A System Simulation Based Approach for Embedded ASIC Design", Proceedings of the On-Chip System Design Conference 1997, Design Supercon, Santa Clara, CA, January 21-23 1997.

Stoaks, P. and P. Sandborn, "Modeling the Impact of Packaging During High-Level System Design", Proceedings of the High-level Electronic System Design Conference 1997, San Jose, CA, October 7-9, 1997.

Vertal, M., "Foresight: System Simulation for System Developers", Proceedings of the 27th Annual Simulation Symposium, IEEE Computer Society Press, La Jolla, CA, April 11 - 14, 1994.

Ward, P. and S. Mellor, *Structured Development for Real-Time Systems,* Yourdon Press, Prentice Hall, New York, 1985.

Waller, L., "The VSI Alliance: Establishing the Standards That Will Drive Tomorrow's Innovations" *Virtual Chip Design (A Supplement to Integrated System Design),* pp. 28-30, May 1997.

Wolf, W., "Hardware-Software Co-Design of Embedded Systems," *Proceedings of the IEEE,* pp. 967-989, No. 7, 1994.

TRADEMARKS

Foresight is a trademark of Nu Thena Systems, Inc.
Seamless CVE is a trademark of Mentor Graphics, Inc.