A Model for the Coanalysis of Hardware and Software Architectures

Fred Rose, Todd Carpenter, Sanjaya Kumar, John Shackleton, Todd Steeves

Honeywell Technology Center
Minneapolis, MN

rose_fred@htc.honeywell.com

Abstract

Successful multiprocessor system design for complex real-time embedded applications requires powerful and comprehensive, yet cost-effective, productive, and maintainable modeling. The multi-disciplinary, VHDL-based modeling library developed by the Honeywell Technology Center places heavy emphasis on multiprocessing and distributed communications. These models focus on detailed hardware performance analysis along with multiple abstraction levels for software representation and evaluation. This paper will detail the processor model which provides the key element for the coanalysis of hardware and software system architectures.

1 Introduction

The Honeywell Performance Model Library (PML) [1][2] was created to fill the analysis needs required for the design of large, distributed, embedded real-time systems. The Rapid Prototyping of Application Specific Signal Processors (RASSP) program [3] is a four and one-half year, $150 million Advanced Research Projects Agency (ARPA)/tri-Service initiative intended to dramatically improve the process by which such complex digital systems, particularly embedded digital signal processors, are designed, manufactured, upgraded, and supported. RASSP seeks an improvement of at least a factor of four in the time required to take a design from concept to fielded prototype or to upgrade an existing design, with similar improvements in design quality and life cycle cost. The motivation for RASSP is the need to provide affordable embedded signal processors for a wide range of DoD systems that are state-of-the-art when they are fielded, rather than when they are first defined.

RASSP program goals target systems requiring multi-hundred node designs. The generic processor model presented here has been designed to model the processors of such systems with sufficient granularity to exercise the system software, runtime, and scheduling, as well as application software ranging from abstract data flow to functional detail. This paper focuses on the processor model and its capabilities to support both abstract and detailed software models. Discussion will include the multi-processor communication and distributed operating system capabilities recently incorporated into the processor model. A high level application of the performance modeling library to the Synthetic Aperture Radar (SAR) RASSP benchmark is discussed, including both the benefits and limitations of such an approach.

The PML is being used by several organizations within the RASSP community, including technology base programs and both RASSP primes. This library is being incorporated into the commercial Performance Modeling Workbench (PMW) 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. Typical design parameters include the entire software design or architecture, system software, and portions of the hardware. Parameterizeable hardware attributes include

- number and types processors at individual nodes.
- memory features
- intranode communication mechanism (bus vs. shared memory)
- internode communication support i.e. custom vs. general processor (shared or dedicated)

RASSP-wide standard interfaces to the performance models have been defined. [4][5]

2 Other codesign approaches

Hardware/software codesign research has taken many forms. There have been attempts to transfer technology between the software and hardware engineering domains [6]. Efforts have focused on the investigation of common modeling representations for hardware and software. Examples include concurrent processes, finite state machines, and unified representations, such as MCC's integrated semantic model [7] and the decomposition graph [8]. Several design environments have been devel-

0-8186-7243-9/96 $05.00

oped to analyze hardware/software systems: SARA [9], SES/Workbench [10], ADAS [11], SIERA [12], and PTOLEMY [13].

A significant amount of effort has been devoted to investigating hardware/software partitioning algorithms. Many of these approaches have been utilized within cosynthesis systems. Two classes of approaches are hardware-oriented [14] and software-oriented [15]. Research [16] has also been performed in the use of clustering techniques to control the partitioning process. More recently, an algorithm [17] which selects an appropriate objective function at each step of the partitioning process has been developed. The selection of an objective function is based on (global) time criticality and local node characteristics, which reflect tendencies towards software or hardware implementation.

The work described in this paper focuses on providing an environment that supports early hardware/software performance evaluation and trade-off exploration. Encompassed in this effort is the development of appropriate abstractions for hardware and software. This environment can also be used for hybrid modeling [18][22], simulating performance models and implementation-level models in a common environment.

The primary differentiator of this work is the attempt to make the modeling library as generic, configurable, and portable as possible. VHDL provides many capabilities to support these features. The only tool requirement for using this library is a fully compliant VHDL simulator. [19] The library has been used with a number of different VHDL simulators.

3 Modeling environment

3.1 Process flow

The PML, and the subsequent commercial PMW, establish a computer assisted environment for analyzing and designing complex systems comprising large numbers of hardware and software components. The PML is comprised of model-based representations of system building blocks. The PML supports an ordered architecture development process targeted at rationalizing architectural feature selection against measurable performance goals. The process combines traditional system decomposition techniques with simulation-based performance experimentation and analysis to support rapid system prototyping in a virtual environment. This environment in turn provides a platform for using predictive performance measurements to analyze and optimize system design trade-offs.

The RASSP approach to system development [20] is based on the spiral system development model which executes in an iterative fashion, successively repeating the requirement-design-test cycle at increasing levels of granularity until a sufficient amount of design detail is revealed to adequately drive error-free implementations. Augmenting the spiral development model with virtual prototyping via simulation-based performance analysis supports rapid iteration through multiple levels of granularity while validating and optimizing a leveled abstraction before proceeding to the next iteration. The result is a reduced development timeframe for producing a validated architecture.

The rapid prototyping facilities central to the performance-based spiral process model are made possible by the reusable PML elements which support rapid assembly and analysis of architectural structures using proven component models.

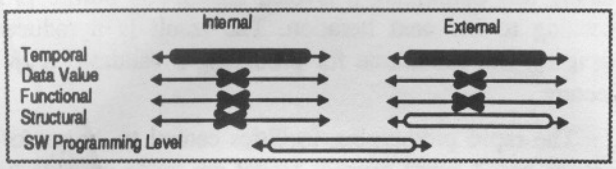
3.2 Performance modeling

Models are evaluated in numerous ways. They may be analyzed, simulated, emulated, or prototyped. The performance models discussed here are primarily simulated. Analytical models can rapidly become too complex to fully represent important system features such as resource contention. Emulation and prototyping are expensive, time-consuming, and necessarily occur late in the design cycle. Simulation provides results that may not be easily attained via analytical models, and faster and at less cost than prototyping. Additionally, simulation can accommodate mixed levels of design and various levels of fidelity and accuracy. However simulation does suffer from significant startup costs, complexity, and significant execution (CPU) times. A robust system design process should utilize all model evaluation methods as appropriate.

The various names associated with modelling abstractions are frequently instance-specific and application-specific, which can lead to confusion. RASSP is no different than other large distributed projects in this respect. A RASSP taxonomy working group [4] has been formed to address this within the RASSP community. For an exhaustive model taxonomy, the reader is referred to Hein, et al. [4] This taxonomy is also available at http://rassp.scra.org/public/atl/taxonomy.html. The following definition of performance modeling is based on Hein's work.

Performance is a collection of the measures of quality of a design relating to the timeliness of the system in reacting to stimuli. Measures associated with performance include response time, throughput, and utilization. A performance model may be written at any level of abstraction. A highly abstract performance model might only resolve the time a multiprocessor cluster requires to perform major system functions, or it can be a less abstract model describing the time required to perform tasks such as

memory access of a single CPU. In the context of RASSP, however, the typical abstraction level of a performance model is often at the multiprocessor network level, also called a network architecture performance model. Internal and external data values are not modeled, except for control information. Figure 1 shows the description of performance models using the RASSP taxonomy.



Symbol Key:

- Model resolves information at specific level relative to Table 1.
- Model resolves information at any of the levels spanned, case dependent.
- Model optionally resolves information at levels spanned.
- Model resolves partial information at levels spanned, such as control but not data values or functionality.
- Model does not contain information on attribute.

Figure 1: RASSP definition of performance models

An important additional benefit of performance modeling is that it provides early interaction of system, hardware, and software designers.

Performance modeling is used during functional analysis, architecture selection and verification, and to a much lesser extent, detailed design. These steps are shown in Figure 2 in the context of the RASSP design process. While performance modeling can be used at all these levels, the particular model presented in this paper is focused on architecture selection and verification. Architectural level includes the actual device or entity under study such as a signal processor, and its environment, such as sensors and actuators. In the case of an electronics system, an architectural level description would include information about both the hardware and software. Note that the definition of "system" is loose here. While we constrain the application of our performance models to electronic systems (although we have performed human==>electronic console operability studies with the performance library [21]), the library is fully capable of representing systems consisting of ASICs, boards, and subsystem cabinets, and sensor networks. This library has also been used in domains outside digital signal processing.

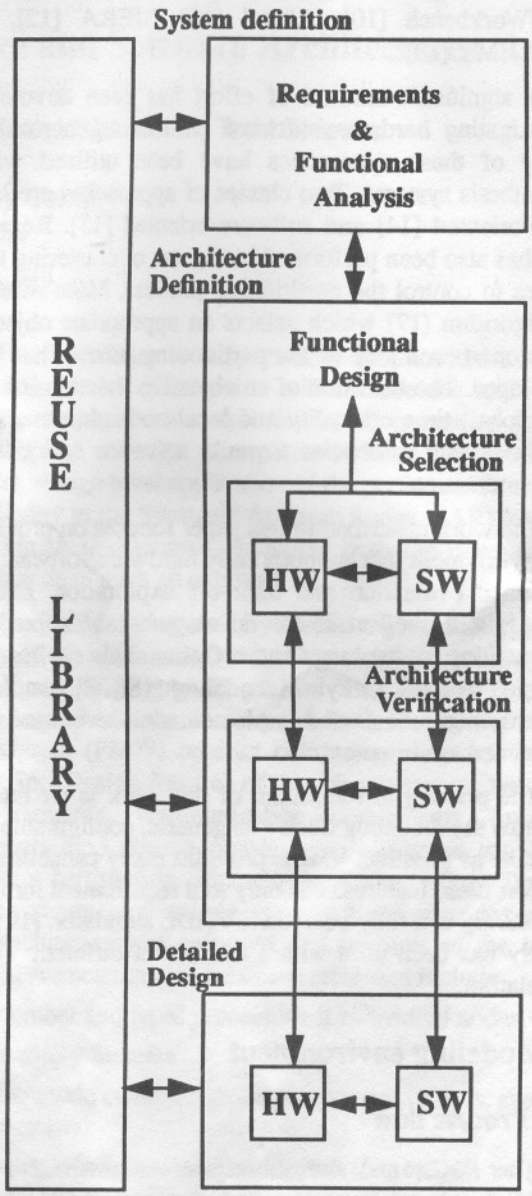


Figure 2: The RASSP design process consists of system definition, architecture definition, and detailed design.

3.3 Library overview

Honeywell has developed the PML in VHDL using standard commercial VHDL capabilities [1]. This allows the systems architect to capture the system under study in a consistent, verifiable form. The library consists of high-level building blocks such as configurable input/output devices, memories, communication elements, and processors. The processor model is the key element to the performance modeling methodology as it facilitates hardware/

software codesign and coanalysis. These building blocks can be rapidly assembled and configured to many degrees of fidelity with minimal effort.

Standard output routines tabulate and graph performance statistics such as utilization and latency. The results can be directly compared with the system specification to verify that the architecture meets the performance requirements. Once the architecture is verified (the latency, utilization, and throughput meet requirements, the system is self consistent, and size, weight, and power limits are met), the system is ready to proceed to detailed design.

As this paper is focused on the particular features of the library that support hardware/software codesign, the reader is referred to [1] for a complete description of the PML and its capabilities.

4 Hardware/software modeling

4.1 Processor model

A critical factor in evaluating performance of complex system architecture is very often accurate characterization of the processor(s) contained in the design; in particular, the capability to model software, if appropriate, with the full detail of actual code executing in the system. In addition, the driving requirement for many design methodologies is the ability to model the highly complex software and hardware level interactions. As a result, much effort was expended to develop a flexible, high fidelity processor model. This model provides a powerful software modeling capability over a wide range of model levels: from modeling of actual code to high level data flow representations. The software modeling is built on top of the full capability of VHDL. Additional features have been added to handle interrupt service, preemptive tasking, task communication, task synchronization and other services one would expect from an executive or general purpose operating system. The scheduling model can also support static and dynamic tasking and even rate monotonic scheduling.

The processor model can automatically provide reports that detail processor task activity timelines, missed deadline reports, processor utilization, task processor utilization profiles, and overall latency.

Figure 3 illustrates two types of processor models; the Light Weight Processor and Multitask Processor models. The Light Weight Processor (LWP) model provides the equivalent of a bare processor. The LWP has less overhead and provides a more efficient software modeling platform for large multiprocessor simulations. All required system services such as communication and scheduling must be supported directly in the software application models.

The MuliTask Processor (MTP) model supports all the

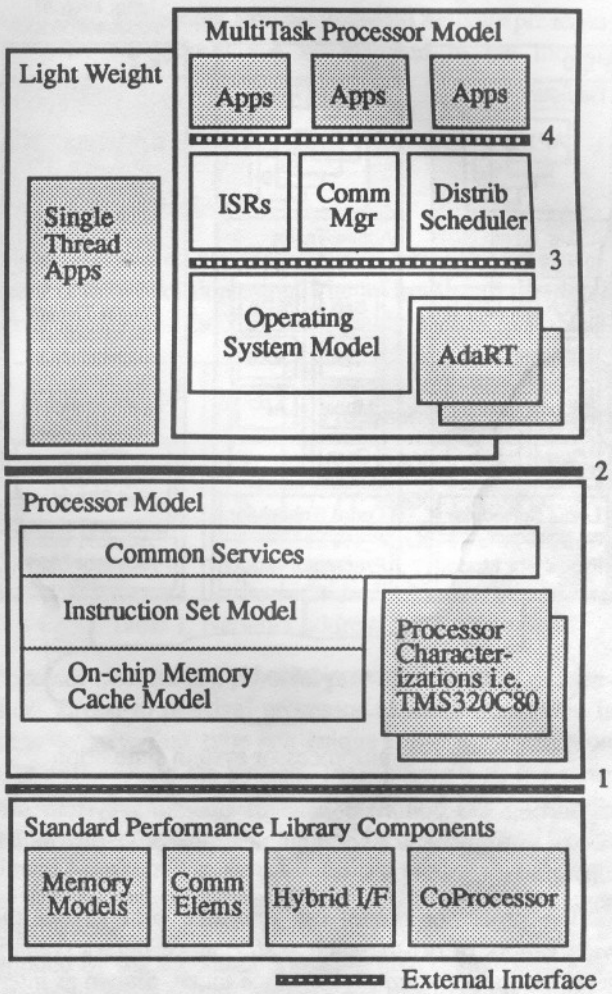


Figure 3: Processor model architecture

services one would expect of a commercial OS as well as a set of distributed multiprocessor OS services. Both forms of the processor model can be used together in a single simulation as shown in Figure 4. This brings the best of both to bear on large multiprocessor applications. The MTP model would be utilized for control and scheduling and the LWP would be used for dedicated applications where scheduling and multiple threads are not necessary.

The multitask processor model contains a well defined uni-processor operating system interface, denoted as interface 3 in Figure 3. The highest level interface, identified as 4, provides the distributed scheduling and communication services. This includes a network addressing, mailbox, and name services capability.

Both processor types are supported by a single core processor model. This core processor model supports the interface, denoted as 2 in Figure 3, which can host single thread applications directly or provides the platform for

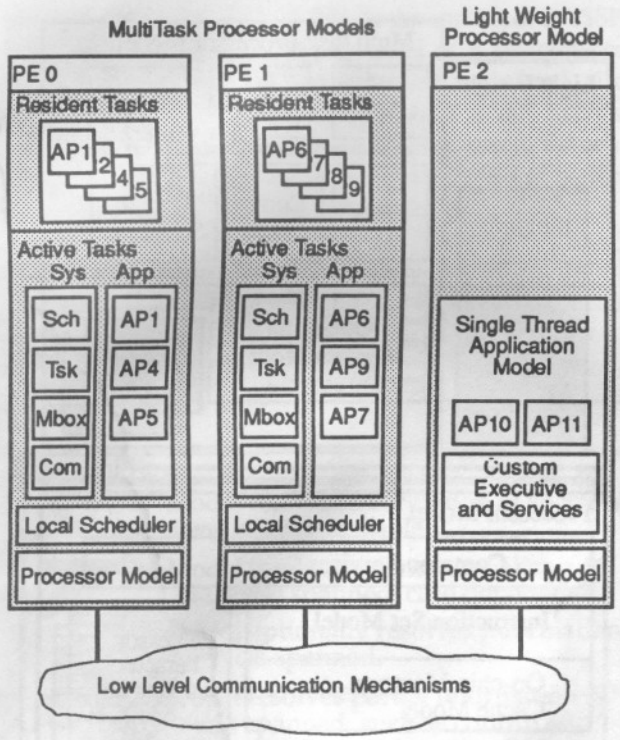


Figure 4: Multiprocessor system simulation

supporting the complete suite of operating system services, multiprocessor communication, and distributed scheduling,

The processor model can interface with any of the wide variety of off-the-shelf performance library components via the standard performance token. Shown as interface 1 in Figure 3, the processor can interact with memories, communication components, and co-processors from the libraries. If necessary the user can supply custom components which utilize token application specific fields. The processor software models can read/write any of the token fields for transfer of application unique data to and from external components.

4.2 Software modeling/representation

In digital systems, software accounts for an increasingly larger portion of the overall system functionality. Often preliminary software designs are never evaluated or simulated against candidate hardware designs. These two design activities are typically completed nearly independently. The designs often don't meet until later in the integration stages. At these later stages any problems encountered by mismatch of the two designs can have serious consequences on either requirements or cost and schedule. The PML, specifically the processor model, provides a means to simulate both hardware and software designs very early in the development process. At this development stage any problems uncovered can be remedied very cost effectively. The processor model provides an essential link between hardware and software that can be used from the very early design phases through detailed design.

The level of hardware model necessary to support software models is quite flexible and need only contain hardware performance level detail. As a minimum, a single processor model can be used with its internal memory/cache model. As multiple processors are added to the design, communication components such as network interfaces and Xbars must be added and the system topology captured. If processor/memory bus contention between multiple bus elements is an issue, the processor memory model must be extended to reference external memory components.

Software models can be developed at various levels, from performance level to complete detailed functional models. An important aspect of the processor model is its ability to support functional software models on a performance level hardware model. If it is necessary for functional software components to interface with functional hardware components, special adaptors known as hybrid interfaces [22] are required. Functional data exchange will occur outside the standard token interface in these cases.

Often one begins with a performance level model of software which is essentially a high level data flow representations of the software architecture. This preliminary model aids in identifying critical areas of the software that will possibly require more detailed model development. The user may decide to functionally model all elements of the software or just the critical areas. In either case, these functional models can be taken all the way down to individual instructions of the final software. The hardware architecture can be evaluated against the evolving software design.

One other possible scenario is when the designer begins with existing algorithms in pseudo-code or another programming language such as C or Ada. The first step in this case will be to translate those algorithms in VHDL, which is uncustomary since VHDL is not often thought of as a language for software or high-level modeling, Nevertheless, VHDL is a full expressive language, so the translation of the algorithms is straight forward.

Once translated the designer has a vehicle for the subsequent detailed design activities. These algorithms can initially be hosted on a very elementary model of the hardware, as simple as a single processor, memory, and communication components. Once these pieces are brought together the hardware and software designs can be performed collaboratively.

The main software design activities will be development of the architecture which include partitioning, allocation, scheduling, communication and possibly some hardware software trade-offs. The hardware activities will include processor evaluation, processor clock rate, processor count, communication network bandwidth and topology.

4.3 Multiprocessor communication

Extending the performance models to a parallel processing environment requires extensions in the physical communication model which relate to the lowest levels of OSI stack, processor model extension to accommodate portions of the network layer extending up through layers 4 to 5, and distributed OS capabilities such as remote task execution. Figure 5 shows how the low communication layers do not provide adequate services for application processes to communicate in a multiprocessing environment.

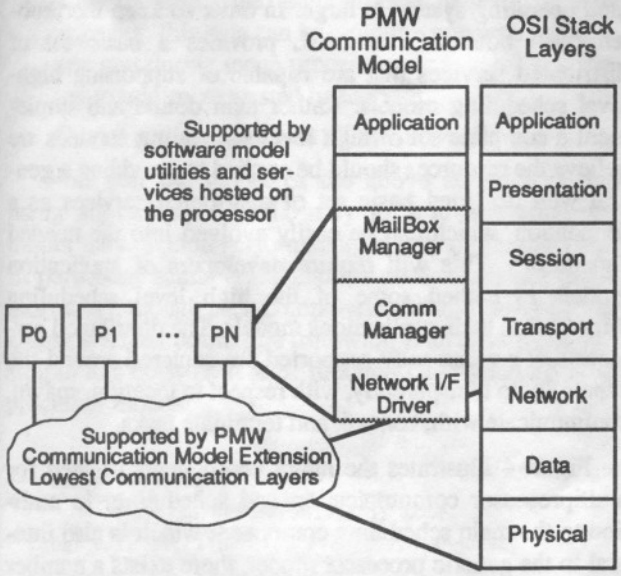


Figure 5: Communication architecture

The communication layers are supported through the communication model extensions. This includes primarily the physical and data layer and a portion of the network layer. Models can be enhanced to support generalized link types, access modes, and operational modes. Access modes include: frequency division multiplex, time division multiplex, and demand access with recovery. Operation modes are primarily circuit switched and packet switched.

Software models hosted on the processor will support services from the network layer and higher. The primary objective of this capability is to provide a process unique mailbox service which is accessible throughout the network. The upper most levels of the OSI such as presentation level will likely not be included in the models capabilities.

4.4 Communication services

4.4.1 Addressing

One of the key elements of multiprocessor communication is network addressing. The address shall consist of a four tuple with the following fields shown in Table 1.

Address Form	Bits	Full Processor	LightWeight Processor
Processor Element	8	1..PE_MAX	1..PE_MAX
Process/Task Type	8	1..TT_MAX	Don't Care
Thread Number	4	1..TN_MAX	Don't Care
Port Number	4	1..PN_MAX	Don't Care

Table 1. Network address structure

Processor element is a physical processor number or identifier. Virtual to physical processor element translation is not necessary. Task type is a unique id for all application tasks available on the system. Two instances of the same task executing on the same processor will have the same first two field of address. They will be distinguished by the Thread Numbers. For dynamic tasks a unique thread number will be assigned for each instance and will no longer be valid once the task has terminated. A unique thread id is supplied by the scheduler when the task is spawned. Rate monotonic tasks will utilize the same thread id for ongoing execution. Port number is an optional field that may be needed to support sophisticated communication structures which could require multiple ports per thread. Tasks may request additional ports which are accessible through the port number field of the address. These additional ports will visible both to internal and external processes. Port number 0 will be the default port which is assigned at task instantiation. The LWP model communication layers do not interpret the task type, thread id, and port number fields of the address but instead pass this information directly along to the single active thread. The task type, thread id, and even port id can be interpreted by the single application thread.

4.4.2 MailBoxes

Every rate monotonic and active dynamic task is assigned a unique event id which operates as a token msg queue or mailbox. These are the same tokens utilized throughout the modeling environment for both hardware and software. The management of event id assignment is

performed by the mailbox manager. The mailbox manager also handles translation of network task address to application task, which includes translation of both event id's and task slots. When messages or tokens are received by the Network I/F, the mailbox manager reads the address and directs the message to the appropriate task. During task initialization, particularly for Rate Monotonic Scheduling (RMS) tasks, the task will register its task type. Dynamics task are registered with the mailbox manager at the time they are spawned.

4.5 Operation

The basic network address operation and structure for a typical processor is shown in Figure 6. Each task thread

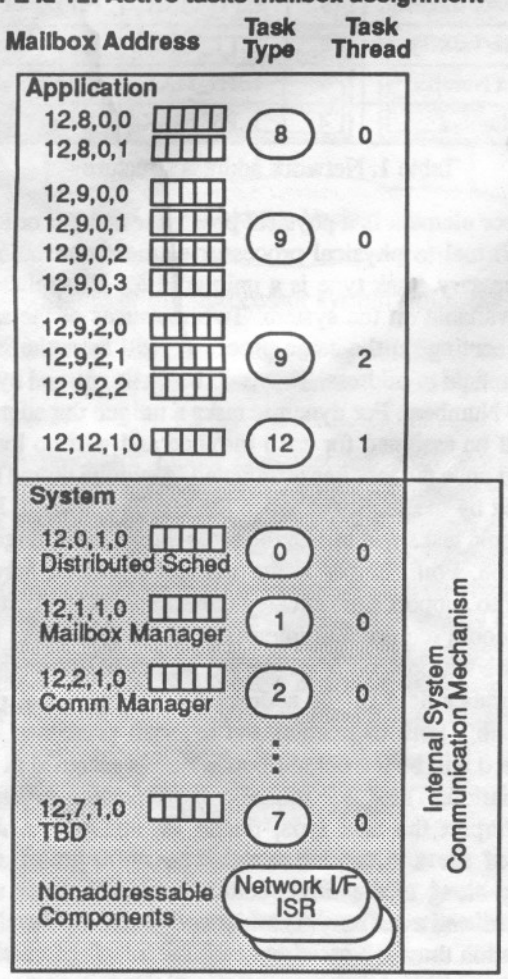


Figure 6: Task network addressing

is assigned a unique mailbox for receiving data. The mailbox manager receives all incoming network messages from the communication manager through a single mailbox. The messages are then sorted and distributed to the appropriate application mailboxes. Messages sent by all local application tasks are sent to a the mailbox managers mailbox. If the mail is local, based on the PE Id check, the address is looked up and the mail sent directly to the mailbox for the local task. If it is not local the mail will be passed down to the communication manager to be sent out on the network

Most of the local interprocess communication will not be supported on the LWP model. Since the LWP supports a single thread any network address with the appropriate PE id will received by the single application thread. The single thread application can interpret these addresses, including task type, thread id, and port number and then send out the appropriate responses. These response can be to other LWP in which case the task type and thread may or may not be need, but any communication to a MTP model will require these fields to be set

4.6 Distributed OS capabilities

The scope of activities required to develop a distributed operating system is large. In order to keep the problem well bounded, the PML provides a basic set of distributed services that are capable of supporting high-level scheduling models. Rather than define and implement a complete set of high level scheduling services we believe the resources should be applied to providing a general well designed basic set of distributed services as a foundation, which can be easily evolved into the needed capabilities. This will require developers of application models to embed some of the high level scheduling directly into their applications models. The distributed services that are currently supported are centered around the capability to transparently, with respect to location, spawn, communicate with, control, and terminate tasks.

Figure 4 illustrates the major components utilized for multiprocessor communication and scheduling. In addition to the main scheduling component which is also integral to the generic processor model, there exists a number of system level components necessary to support the high-level services of a true multiprocessing environment. Included here are scheduling and communication components that handle most of the needed distributed services. The lightweight processor model, illustrated as PE 2 in the figure, can be used interchangeably with the multitask processor model.

5 Results/examples

The RASSP Benchmark One focuses on a SAR design [23]. We constructed a performance model using the PML, based on the SAR performance model handcrafted by the Lockheed-Martin Advanced Technology Laboratory (ATL) RASSP team [24]. The intent of our modeling activity was multifold:

- Demonstrate how the PML can be used to model a large distributed application
- Compare process, model, and results to the custom performance modeling approach.

The model we developed has two primary constituents, the hardware and software. One of the features of the PML is that the distinction between the two can be clearly, and easily made. We modeled a critical computational portion of the SAR algorithm, consisting of the three following code modules:

- Input Module - Distributes range vectors to the compute processes.
- Compute process - (1) Receives a range vector, computes range compression, and then sends corner turn data to its neighbors; (2) it registers the corner turn data it received from the other compute nodes, performs the corner turn; (3) the process computes azimuth compression on the corner turn vectors; and (4) sends the data to the output process. Depending on the mapping, this data can be from the previous frame, and sent during range processing or azimuth processing to minimize wasted bandwidth.
- Output process - collects all the output data.

Note that the model of the above software is done using algorithmic VHDL, only to a level sufficient to model the expected performance aspects of the algorithm. In particular, actual computations on data are not part of this model, though such computations could easily be integrated at a later point. The following is an example of how simple this code is. It uses function calls standard to the processor model:

```
--  Range Processing
RangeProcessing: FOR pulse IN 0 TO
        ((NPulse/NPEsPerPolar)-1) LOOP
  -- Receive in-coming data.
  local_wait_until(pending_events,ProcRequest,
        ProcReply,INPUT_EVENT);
  -- Compute the Range-pulse.
  execute (ProcRequest, ProcReply, FLOP,
         NumRangeInst*(WordSize/ word_32)*NSamples,
         INST, DEBUG);
  compute_target := pid;
  --  Send the Corner-turn data
  SendCornerTurn: FOR k IN 1 TO NPEsPerPolar-1 LOOP
    compute_target := next_compute_pid (pid,
                  compute_target);
    send_data_event (ProcRequest, ProcReply,
                  CORNER_EVENT,
                   routePid, compute_target);
                   WordSize*NRange/NPEsPerPolar,
                   CORNERPUT_PRIORITY,
                   INST, DEBUG);
  END LOOP SendCornerTurn;
END LOOP RangeProcessing;
```

The other major part of the model is the hardware architecture, and that breaks down into the primary pieces of the network and the processors. We are using the lightweight processor model, since each SAR processing node is basically single threaded, and the capabilities of our scheduler is unnecessary for this model.

The communications network is an interconnected fabric of Mercury Raceway crossbar components, arranged in a tree configuration specified by ATL which is shown in Figure 7. This example uses several specialized communi-

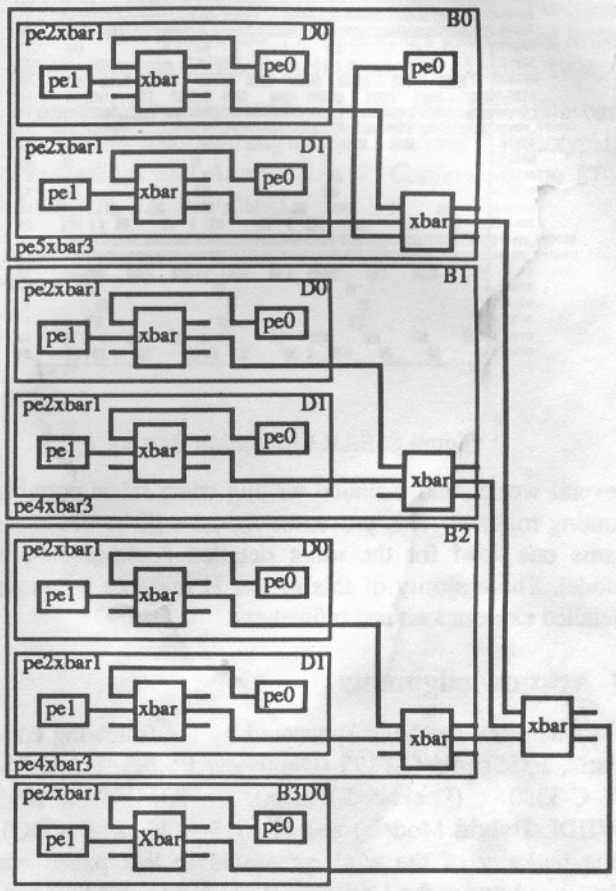


Figure 7: Mercury raceway SAR topology

cation components to model the 6 port crossbar, which measures the throughput and latency, and also accounts for message retries and preemption.

Figure 8 shows a slice of the simulation results. It is an activity plot, showing when hardware and software components are active in the system. Time is on the horizontal, and is in nanoseconds. Objects of interest are on the vertical, and in this case show paths through the crossbars, as well as processor activity. Some of the crossbars are fairly constantly loaded, while others are lightly loaded. Numbers on the right indicate utilization of those components.

6 Summary

The SAR example is intended to demonstrate how coarse grained, abstract models could be rapidly developed using the PML. This example also points out the coanalysis features of the processor model. The processor model and software was developed and debugged over

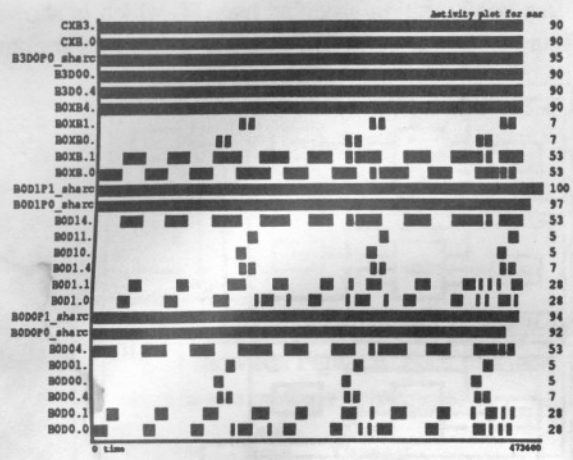


Figure 8: SAR Simulation Results

several weeks, and included writing some fairly portable routing routines. This processor model will be the exact same one used for the more detailed communications model. The majority of this model is reusable for more detailed explorations and refinement.

7 Acknowledgments

This work has been sponsored by the following contracts, F33615-94-C-1495 (Omniview PMW), DAAL01-93-C-3380 (Lockheed-Martin), F33615-94-C-1495 (VHDL Hybrid Models) and F33615-92-C-3802 (CDG). The majority of the work presented in this paper was developed under the Lockheed Martin ATL RASSP program.

8 Reference

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

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

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

[4] Hein, C., et al, ""RASSP VHDL Modeling Terminology and Taxonomy - Revision 1.0," *Proceedings 2nd Annual RASSP Conference*, pp 273-281, Arlington, VA, July, 1995.

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

[6] Smith, C. U., R. R. Gross, "Technology Transfer between VLSI Design and Software Engineering: CAD Tools and Design Methodologies," *Proceedings of the IEEE*, Vol. 74, No. 6, June 1986, pp. 875-885.

[7] Franke, D. W., M. K. Purvis, "An Overview of Hardware/Software Codesign," *International Symposium on Circuits & Systems*, May 1992, pp. 2665-2668.

[8] Kumar, S., J. H. Aylor, B. W. Johnson, W. A. Wulf, *The Codesign of Embedded Systems: A Unified Hardware/Software Representation*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[9] Estrin, G., R. S. Fenchel, R. R. Razouk, M. K. Vernon, "SARA: Modeling, Analysis, and Simulation Support for Design of Concurrent Systems," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, February 1986, p. 293-311.

[10] Scientific Engineering Software, Inc. *SES/ Workbench User's Guide*, Austin, Texas, April 1989.

[11] Frank, G. A., et al. "An Architecture Design and Assessment System for Software/Hardware Codesign", *Proceedings 22nd Design Automation Conference*, 1985, pp. 417-424.

[12] Srivastava, M. B., R. W. Broderson. "Rapid-Prototyping of Hardware and Software in a Unified Framework", *Proceedings of the International Conference on Computer-Aided Design*, 1991, pp. 152-155.

[13] Kalavade, A., E. A. Lee, "A Hardware-Software Codesign Methodology for DSP Applications," *IEEE Design and Test*, September 1993, pp. 16-28.

[14] Gupta, R. K., G. De Micheli, "Hardware-Software Cosynthesis for Digital Systems," *IEEE Design and Test*, September 1993, pp. 29-40.

[15] Ernst, R., J. Henkel, T. Benner, "Hardware-Software Cosynthesis for Microcontrollers," *IEEE Design and Test*, December 1993, pp. 64-75.

[16] Barros, E., W. Rosenstiel, "A Method for Hardware/Software Partitioning," *Proceedings Compeuro*, IEEE CS Press, 1992.

[17] Kalavade, A., E. Lee, "A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/ Software Partitioning Problem," *3rd International Workshop on Hardware/Software Codesign*, Grenoble, France, September 22-24, 1994, pp. 42-48.

[18] 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, N. J., 1992.

[19] IEEE Standard VHDL Language Reference

Manual, IEEE Std 1076-1993, IEEE Customer Service, 445 Hoes Lane, PO Box 1331, Piscataway, New Jersey 08855-1331.

[20] Pridmore, J., and W. Schaming, "RASSP Methodology Overview," *Proceedings 1st Annual RASSP Conference*, pp 71-85, Arlington, VA, August, 1994.

[21] Carpenter, T., and C. Miller, "Modeling Human Factors with VHDL," *Proceedings Fall 1993 VHDL International Users Forum*, pp 65-75, San Jose, CA, October, 1993.

[22] 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*, pp 147-154, Arlington, VA, July, 1995.

[23] Zuerndorfer, B., and G. Shaw, "SAR Processing for RASSP Application", *Proceedings 1st Annual RASSP Conference*, pp 253-268, Arlington, VA, August, 1994.

[24] Hein, C., and D. Nasoff, "VHDL-based Performance Modeling and Virtual Prototyping", *Proceedings 2nd Annual RASSP Conference*, pp 87-94, Arlington, VA, July, 1995.