RASSP Token-based Performance Modeling Appnote

Abstract

Efficient design of complex systems requires modeling at multiple abstraction levels. This RASSP application note describes token-based performance modeling at an abstract level and how to use it within a rapid-prototyping environment. It describes rapid-prototyping methods developed and demonstrated on the RASSP project that improve simulation speed by several orders of magnitude.

Purpose

This application note should be read by system architects, software designers who are responsible for deciding how to partition software among multiple computer nodes, and hardware designers who are responsible for selecting appropriate network configurations and components. The basic concepts can be applied to other systems and other levels of abstraction.

Roadmap

RASSP Token-based Performance Modeling Application Note

1.0 Introduction

RASSP application notes augment course modules and case studies about digital system design. The material is applicable to the design of complex systems such as digital signal processing (DSP) or control systems and other multiprocessor systems. The application notes serve to document the design methods that were developed on the RASSP program. This application note describes the purposes and methods for token-based performance modeling. Additional details regarding processor types, model hierarchy and abstractions, common language and vocabulary can be found in the VHDL Terminology and Taxonomy document.

1.1 What is Token-based Performance Modeling

Computational Performance is a collection of measures relating to the timeliness of a system design in reacting to stimuli. Measures associated with performance include response time, throughput, and utilization. The generic term performance model refers to a model of any abstraction level that describes the timing relationships without resolving non-time-related aspects, i.e. values, formats or functions.

A highly abstract performance model could resolve the time consumed by cluster of processors to perform major system functions like search, FFT or FIR. A less abstract performance model could describe the time required to perform detailed tasks such as a single CPU memory access. In the context of Lockheed Martin Advanced Technology Laboratories (ATL) RASSP system design, the typical abstraction level of a token-based performance model is at the multiprocessor network level; sometimes called a network architecture performance model.

Token-based performance modeling is defined in the RASSP Taxonomy as a performance model of a system's architecture that represents data transfers abstractly as a set of simple symbols called tokens. Neither the actual application data nor the transforms on it are described other than that required to control the sequence of events in time. The application-data is not modeled, while only the control-information is modeled. For example: answer = 5 is not modeled, but control node = search is modeled.

Typically, the token-based performance model resolves the time for a multiprocessor networked system to perform major system functions. It keeps track of the usage of resources such as: memory buffer space, communication linkages, and processor units. The structure of the network is described down to the network node level. The network nodes include processor elements, network switches, shared memories, and I/O units. The internal structure of the network nodes is not described in a Token Based Performance Model.

1.2 Why and when is Token-based Performance Modeling needed

The growing reliance on commercial-off-the-shelf (COTS) processing components and state-of-the-art comprehensive simulations of the internal logic of individual integrated circuits (ICs) have reduced the number of design errors to the point where the majority of design faults now occur in the component requirements specification. It is now becoming expedient to focus on the system verification process, where the IC requirements are developed.

Digital systems are growing ever more complex with improving technology and integration densities. In particular, multiple processor elements are harnessed to extend a system's processing power beyond that of single processor technology. Conventional design methods such as physical prototyping or gate level simulation, become prohibitively costly and time consuming for such systems.

Unlike the testing of an individual IC design which typically requires on the order of thousands of clock-cycles or a fraction of a second of simulated time, digital system simulation typically requires the simulation of significant portions of an application algorithm spanning several seconds or minutes of simulated time. The simulation of multiple cooperating Processing Elements (PEs), each executing many millions of instruction cycles per second (MIPS), over such time spans represents the execution of an extraordinarily large number of events. Therefore, simulation of a multi-processor system at or below a chip-behavior level becomes impractical due to the large memory and simulation run-time that would be needed.

However, full-system simulations are required to validate the overall system concept; to jointly optimize the hardware and software; and to investigate the interactions between cooperating subsystems prior to embarking on time-consuming and detailed designs that could unknowingly be misguided. Fortunately, the number of events can be reduced by adroitly using abstractions without jeopardizing the accuracy or validity of the model. Therefore, selecting an appropriate modeling abstraction level is crucial for timely design.

Simulations must execute and return results from simulation runs in roughly an hour or less to permit designers to rapidly explore, optimize, and verify system design solutions. Such a turn-around time allows several design iterations per day and ensures convergence on a valid system design in a matter of days instead of months. Therefore, more abstract prototyping methods are needed for verifying all functional aspects of a complete integrated system early in the design process.

Token-based performance modeling is an abstract form of system modeling that addresses these challenges. The simulations help identify bottlenecks, validate early system expectations or coarse timing requirements, and highlight initial design options.

1.3 How it fits into overall design process

Token-based performance modeling supports Lockheed Martin ATL's System Design and Architecture Design process as shown in Figure 1 - 1.

As the overall system requirements are established by the System Design process, the architecture design process decomposes the overall requirements into a set of requirements for the constituent building-blocks that together satisfy the overall system requirements. Figure 1 - 1 shows the relationship of performance modeling to the overall design process.. The architectural building-blocks consist of software tasks, processing elements, memories, I/O units, and a network that connects them together.

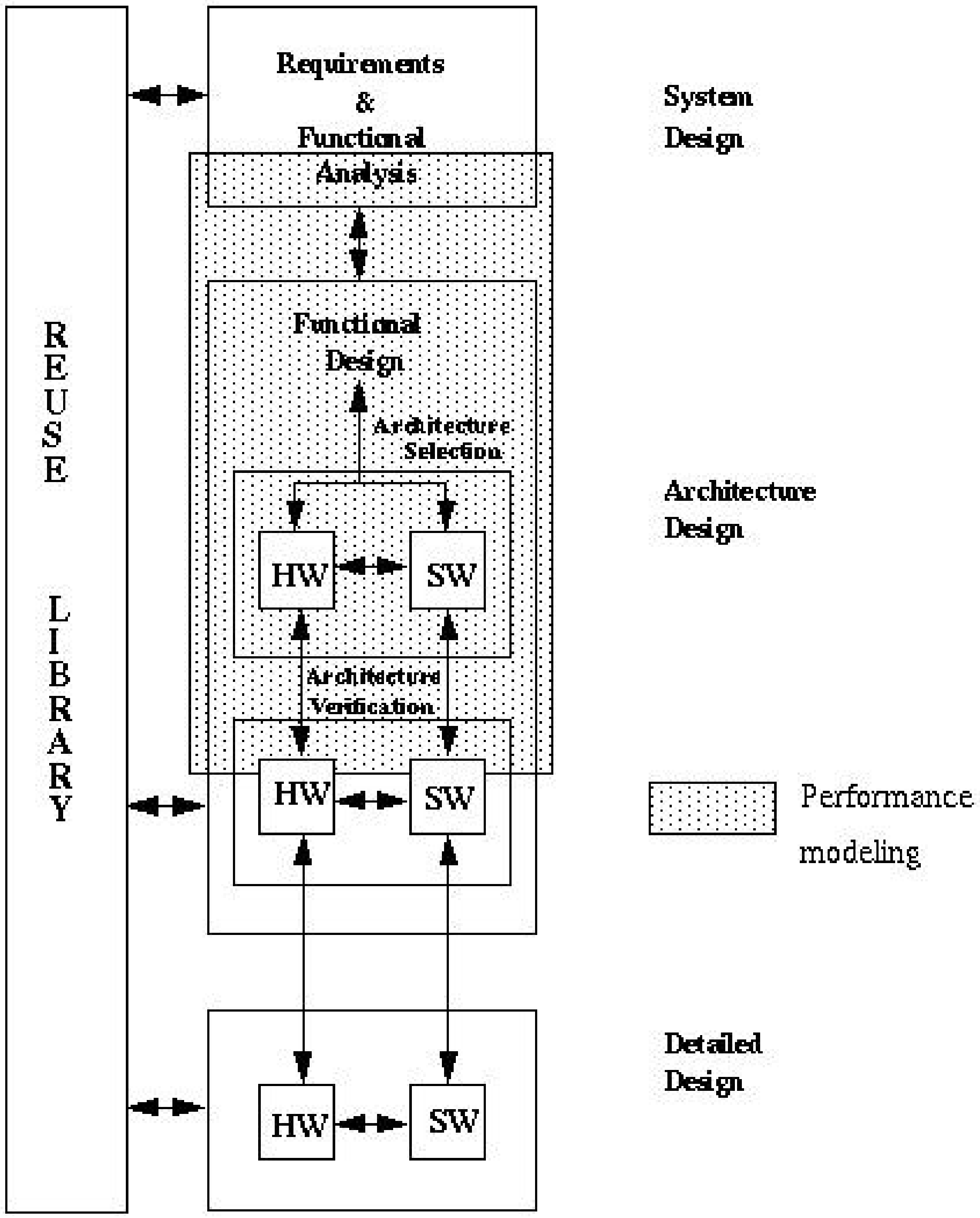


Figure 1 - 1: The RASSP design process consists of system definition, architecture definition, and detailed design. The shaded area indicates where token-based performance modeling occurs.

The architecture design process proposes candidate solutions that consist of combinations of specified processor elements, linkages, memories, network configurations, and application-software mappings. The components are specified in terms of their relevant performance ratings. The token-based performance model is used to test how well various candidate solution combinations meet the overall requirements.

Feed-back from the token-based performance model is used to optimize the architectural selections and eventually to select the best architecture for meeting the given requirements. As a result of the performance modeling, the selected architecture is specified in terms of performance requirements for each of its constituent building-blocks.

The requirements for each building-block from the architectural specification are passed down to the detailed hardware/software design processes as shown in figure 1 - 2. As the component designs are realized or acquired for testing during the detailed design process, the actual parameters of the component's specification become known to a higher degree of confidence and accuracy. In some cases, the proposed requirements for a component cannot be met. In other cases, the requirement can be met or exceeded more easily than expected. In either case, the resolved specifications from the detailed design process are periodically injected back up to the architecture design process's token-based performance model to investigate component requirement reallocations and to ensure that the emerging design still meets the overall requirements.

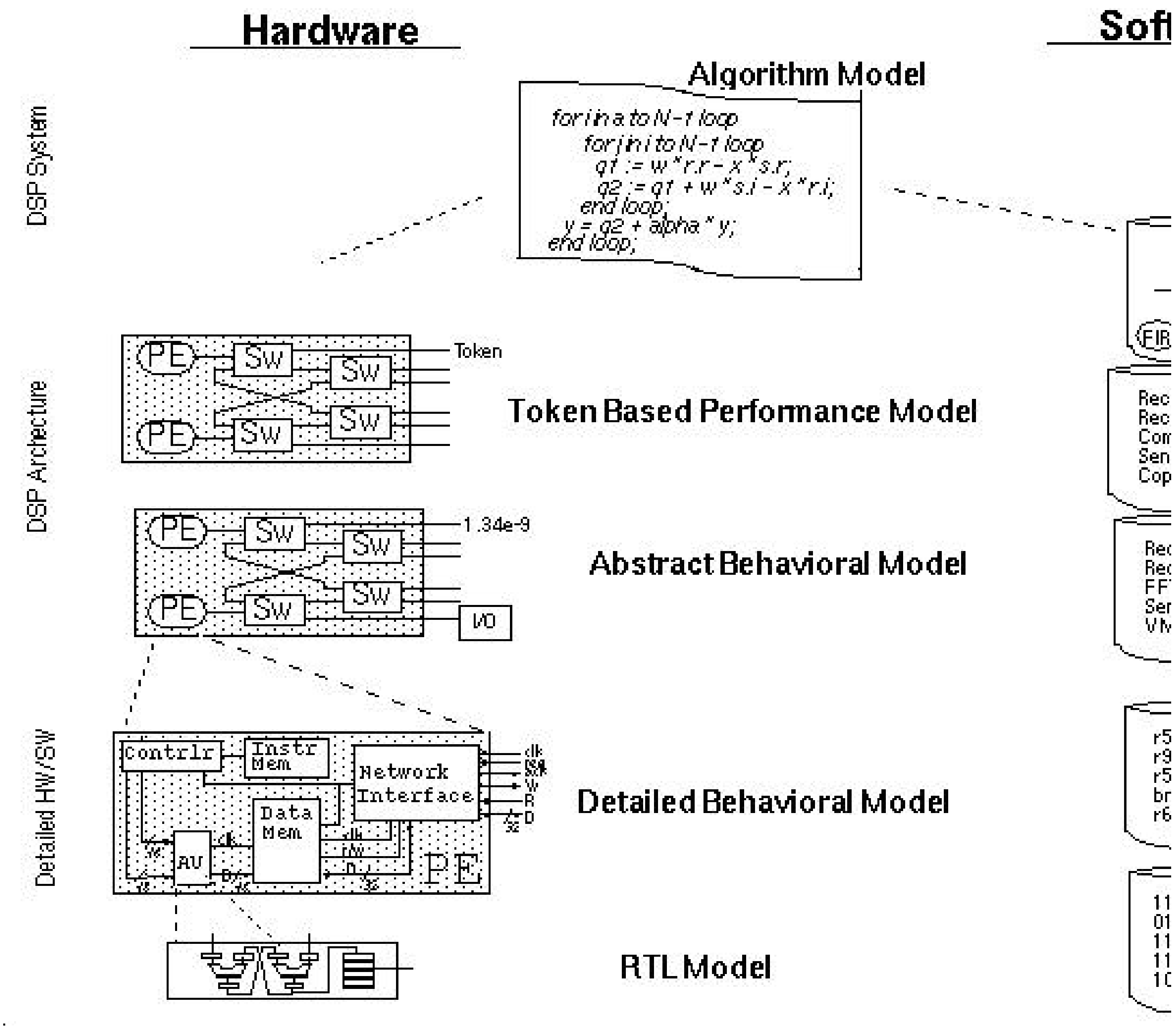


Figure 1 - 2: Levels of the Design Process.

Next Up Previous Contents
Next: 2 Introduction **Up:** Appnotes Index **Previous:** Appnote TOKEN Index

RASSP Token-based Performance Modeling Application Note

2.0 Purposes

The primary purposes of a token-based performance model are to determine the sizes, architecture, software-to-hardware mapping, and performance requirements for each of the major components of the system. Specifically, it determines the sufficiency of the following selections in meeting the system processing throughput and latency requirements:

- number and type of processor elements,
- the size of memories and buffers,
- network topology (bus, ring, mesh, cube, tree, or custom configuration),
- network bandwidths and protocols,
- application partitioning,
- mapping, and scheduling of tasks onto processor elements,
- flow control schemes.

The total processing latency, throughput, and physical constraints on the processing system drive the optimization of the processing architecture in terms of the number and type of processing elements, and the memory and buffer requirements.

The network architecture specifies the topology of the network, as well as the bandwidth and protocol requirements for the individual network links. Typical network topology families for digital processing systems include: bus, ring, mesh, cube, tree, and custom configurations. Topologies are selected to match the particular data transfer patterns of the specific application. Performance simulations provide information concerning link and processor utilization measurements for specific topology, processor element (PE) type and software mapping combinations.

The software-to-hardware mapping divides the application algorithm into separate tasks, which are allocated to the individual PEs. The tasks are then scheduled according to their relative data dependencies and the overall processing latency constraints. Various combinations of mapping and scheduling methods can be tested and selected. For example, depending on the application, the mapping and scheduling can be assigned at design-time (i.e. statically) or at run-time (i.e. dynamically).

There is no efficient, general, closed-form, optimum solution to the partitioning, mapping, and scheduling problem. It is an not-polynomial-(NP)-complete problem. Consequently, many iterative, heuristic, and manual techniques are currently applied. The performance models facilitate these methods, as shown in figure 2 - 1.

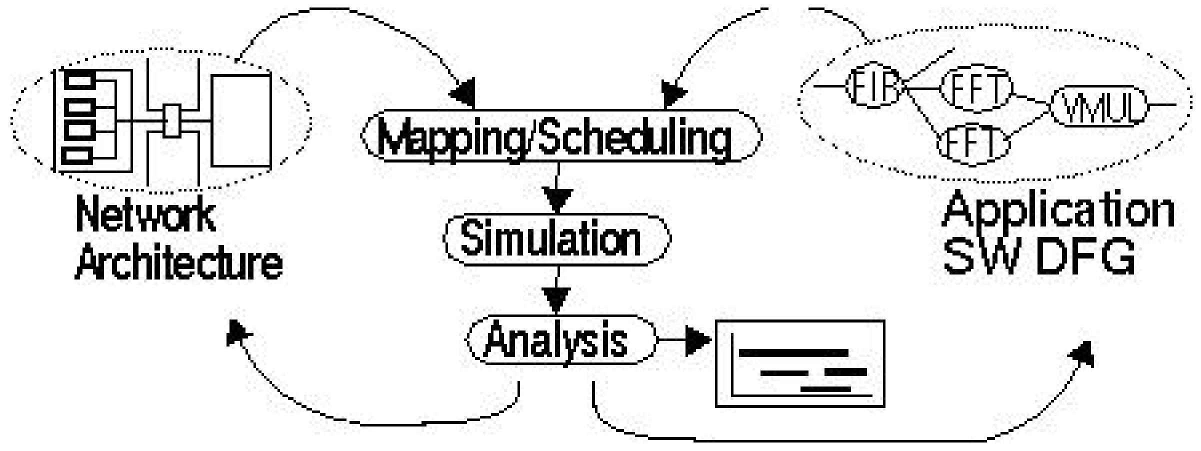


Figure 2 - 1: The Hardware/Software Co-Design Process

2.1 Starting Information

Upon initiating performance modeling activities within the architecture design process, the designer bases selections on the following requirement information that is derived from the system design process:

- Mathematical application algorithm to implement.
- Minimum allowable processing throughput (rate).
- Maximum allowable processing latency (delay from input to output).
- Maximum allowable power consumption.
- Maximum allowable volume (or maximum HxWxL dimensions).
- Maximum allowable weight.
- Input/Output/Control interface specifications.

Subject to the following goals:

- Minimize development risk, cost, and time.
- Minimize life-cycle costs. (Includes consideration of maintainability, repairability, reliability, expandability, supportability, and etc..)

The information above is the information that the performance modeling design activity is based upon and uses as input to the process.

2.2 Results Information

Token-based Performance Modeling is conducted to support the Architecture Design process. The goal of the Architecture Design process is to select an architecture that best satisfies the criteria of section 2.1 above. In this role, the performance model is used to evaluate architecture candidates to determine the :

- Overall processing throughput
- Overall processing latency
- Resource Utilization (processor, memory, link)

An architecture is the combination of processing element types as distinguished by their unique processing rates, memory sizes and locations, and network configuration, bandwidth, and protocol types. There are many potential architectural combinations, and there are many permutations for mapping the application algorithm onto each architecture. The performance model helps the designer understand the impact of architectural/mapping decisions and helps develop strategies for quickly arriving at an optimal solution. The following output from performance models guides the development process:

- Time-line graphs of processing, idleness, and communication activities.

- Utilization statistics for links, processors, and memories.
- Load balancing statistics.
- Identification of bottlenecks and critical paths of processing.

When the designer selects the final architecture and mapping, the performance model will have validated the solution in terms of its overall processing throughput and latency.

Next Up Previous Contents
Next: 3 Metrics **Up:** Appnotes Index **Previous:**1 Introduction

RASSP Token-based Performance Modeling Application Note

3.0 Metrics

The primary measurements associated with token-based performance modeling are throughput, latency, and utilization. Time-line activity graphs display the concurrency of events as a function of time. Related measurements include efficiency which can be derived from the primary measurements as a ratio compared to their theoretical maximum values.

3.1 Throughput

Throughput is the rate at which data is processed by the system. It is usually specified in terms of the number of data-elements or operations that can be processed per unit time. Throughput may be specified as an instantaneous peak or a sustainable average over time. The latter is often more meaningful in determining system size requirements if there is sufficient memory to spread the processing load over time.

3.2 Latency

Latency is the time delay between when an activity starts and when it completes or reaches an equivalent point. Latencies typically apply to processing or data-transfer activities. For instance, a processing latency is the time between when a piece of data is applied to a processing system and when the result due to that piece of data becomes available at the output of the processing system. Or for a communication latency, the latency for a packet transfer is the time delay between when the first word of data transfers from the source device until that first word arrives at the destination device. Note that this definition does not consider the time to transfer the entire packet, as that would be a function of the packet length and the transfer rate (throughput). By defining the metrics in this way, the delays and rates are orthogonal; so latency and throughput are specified separately.

3.3 Utilization

Utilization is the percentage of time that a resource is actively performing an application activity. The opposite of utilization is idleness. Utilization percentages are typically calculated for critical system resources such as processor elements and network linkages or buses.

The instantaneous utilization of a memory element may also be recorded or plotted. Such a measurement is not time-based, but rather expresses the percentage of memory-space consumed or allocated to application data at a given instant in time. The peak value is typically most relevant, and the instantaneous value can be plotted as a function of time.

To calculate a time-based utilization, the total time that a resource is in-use must be accumulated over some time interval. The interval is often assumed from time-zero until several iterations of the algorithm processing have completed to achieve a good steady-state value. Such a time-interval may include some dead-time, as systems can take a while for the data to get distributed to the devices and to reach steady-state processing. If only a few sets of data are applied to the system and the simulation is run until all the data has been processed, then on the trailing-end, many of the devices have long-finished by the time the last device finishes. These effects produce lower utilization percentages than might be expected intuitively.

Alternatively, some designers specify the interval from the first time a given device was used until the last time it was used. However, even this definition can yield misleading results. For instance, when a device is used only once, for only for a very small portion of the total simulation duration, this method would report 100% - utilization even though the device was hardly used.

Therefore, it is recommended that the method for defining the utilization interval be stated and understood while analyzing time-based utilization numbers.

3.4 Time-lines

An activity time-line graph displays the concurrent activity of a set of system resources as a function of time. Typically, the times when a processor is actively computing an application task are shown as a dark or colored bar. Colors often identify the particular software tasks being executed. The intervening times, when a processor is idle while waiting for input data to arrive or while waiting for output data to be sent away, are shown as white-space. Such a plot is specifically called a processing time-line. Processing time-lines provide graphic visualization that shows the designer the patterns of processing concurrency in the design. It also helps reveal bottle-necks and the critical processing paths.

In a similar way, the data transfer activity across the network linkages and buses can be displayed on a time-line graph. Such a plot is specifically called a communications time-line. Communication time-lines help the designer visualize the traffic flow through the system as a function of time. They also help to identify hot-spots or bottlenecks in the architecture or mapping. Such visualization assists in evaluating architectures and guides the designer in balancing the processing load and transfer patterns to achieve the best processing yield from a given architecture.

Although primarily used to display hardware activity, time-lines can also display the activity of software tasks as a function of time. Such a graph is called a software time-line. Colors can be used to identify the particular processor a task is executing on.

Next Up Previous Contents

Next: 4 Performance Modeling Environments **Up:** Appnotes Index **Previous:**2 Purposes

Approved for Public Release; Distribution Unlimited *Dennis Basara*

RASSP Token-based Performance Modeling Application Note

4.0 Performance Modeling Environments

A modeling environment is characterized by the language(s) used to describe the system under design as well as the tools used to work with the models.

4.1 Requirements for a Performance Modeling Environment

The modeling language must possess convenient methods for expressing the structure, functionality, and temporal concurrency aspects of a complex systems. Ideally, the descriptive constructs should be standard to support model interoperability and re-use.

The modeling environment must provide a means to conveniently explore hardware/ software interaction. To facilitate such interaction, the hardware must be described independently from the software, such that a given hardware architecture can be directed to execute a variety of distinct application programs without modification to the hardware model. Conversely, a given software application program should be executable on a variety of candidate hardware configurations.

The network topology must be specified independently from the behavioral models of the network hardware components. Additionally, the component models must be modular so that network components, such as processor elements, can be interchanged without redesigning the network or the component behavior models.

An important feature for the simulation models at the performance level is extensibility to greater levels of detail to support the subsequent design stages. For instance, there must be a means to add functionality to produce abstract behavioral model based virtual prototypes, and synthesizable components.

Another important aspect is the model's efficiency, such that it offers quick simulations that permit the rapid exploration of many design alternatives. The modeling environment must inject minimal overhead such that a significant duration of arbitrarily complex systems and functionality can be simulated.

4.2 Proprietary Language Based Environments

Several excellent performance modeling tools exist on the open market. For example, BONES is a Block Oriented Network Simulation environment produced by the Alta Group Of Cadence Design Systems. Workbench is a system simulation environment produced by Scientific and Engineering Software (SES). Foresight is a system design environment produced by Nu Thena Systems. Each of these tools provide libraries and convenient environments for abstract network architecture simulations.

In BONES, the primary method for describing functionality and structure is through the interconnection of blocks in a graphical paradigm. SES Workbench and Nuthena Foresight promote C and Pascal-like language based methods for describing functionality while providing graphical means for describing topological information.

Each of these companies is the sole definer of language features and provider of tools for their respective environment and language variant. These products tend to focus on the system design issues and do not emphasize linkage to the detailed hardware and software design layers.

4.3 Open-Standard Languages

Standard languages such as (VHSIC Hardware Description Language) VHDL, Verilog, C, C++, or Java offer great potential for integration with other design models that are already based on these languages. However, conventions for their usage in performance modeling have not been previously developed.

The most general purpose languages, such as C or C++, do not contain standard notational constructs for topological structure or concurrent time delays. Although such concepts can be implemented with some added complexity and awkwardness in these languages, the lack of standard methods still leaves each implementor applying incompatible solutions. Additionally, the general purpose languages provide no inherent means to transfer design information to the lower, more detailed, design layers such as RTL.

In contrast, VHDL contains standard constructs for describing topological structure, concurrent time delays, and arbitrarily abstract functionality IEEE Standard VHDL Language Reference Manual. The description of structure and functionality is inherently independent in VHDL.

Verilog is a competing hardware language with VHDL. Although Verilog contains standard methods for describing topological structure and concurrent time-delays, it is specifically aimed at the detailed (Gate Level) hardware design task. Verilog does not possess as powerful mechanisms for spanning the higher abstraction levels such as arbitrary compound data types or custom signal resolution functions that are needed when not modeling pure electrical values.

Of the standard languages, VHDL is uniquely capable of spanning the necessary abstraction layers: from mathematical algorithms down to RTL and logic. It can provide a direct coupling and transfer of design information between the levels.

VHDL is a stable IEEE and ANSI standard language for which a diversified array of vendors offer compilers, simulators, and model libraries. For the above reasons, VHDL became the language of choice for implementing performance models on the RASSP program.

4.4 Open-Standard Language Based Environments

Several approaches to standard language-based performance modeling have been developed under the RASSP program.

- The Performance Modeling Library (PML), and the PML-Library based COSMOS (formerly Performance Modeling Workbench - PMW) were developed by the Honeywell Technology Center (HTC) and Omniview respectively.
- Advanced Design Environment Prototype Tool (ADEPT), was developed at the University of Virginia (UVa).
- The Lockheed Martin Advanced Technology Laboratories Library (ATL-Lib).

All three approaches establish a computer assisted environment for analyzing and designing complex systems comprising large numbers of hardware and software components. All are composed of model-based representations of system building blocks. All three support an ordered architecture development process targeted at rationalizing architectural feature selection against measurable performance goals. This process combines traditional system decomposition techniques with simulation-based performance experimentation and analysis to support rapid system prototyping in a virtual environment.

The ADEPT environment describes systems in a very abstract way in terms of servers and queues. It forms a well-matched tool for queuing system investigations. ADEPT's primitive library is very versatile and can be used in a variety of ways to describe the performance related aspects of a digital processing system or any of its computational devices.

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 (CPN) representation, which is based on Jensen's CPN model. The modules communicate by exchanging tokens, which represent the presence of information, using a uniform, well defined handshaking protocol. 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. A more detailed description of the entire ADEPT module set can be found in the following:

- The codesign of Embedded Systems: A United Hardware/Software Representation
- A Generalized Timed Petrinet for Perfomance Analysis
- Modeling a Real-Time Multitasking System in a Timed PQ Net

The PML and ATL libraries operate on a different paradigm level than ADEPT. They can be considered to be specific to the task of network architecture design. In comparison to ADEPT, their main primitives consist of elements representing complete network components, such as processor elements, network switches, and buffer-memories.

Honeywell has developed the PML in VHDL using standard commercial VHDL capabilities. See the following papers for a more thorough discussion: VHDL Performance Models, Evaluating Distributed Multiprocessor Designs, Advanced Multiprocessor System Modeling. 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 co-analysis. 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. These statistics can be used for performance verification studies.

The differences between the Honeywell PML and the UVa ADEPT can be traced to two 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 mixed-level or abstract-behavioral modeling. This difference can be 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 buses 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.

The performance model library developed by ATL and described in a paper, VHDL-based Performance Modeling and Virtual Prototyping is similar to the PML. The ATL models use the Processor-Memory-Switch (PMS) paradigm to describe the architecture of digital systems. As in the PML, separate representations for hardware and software models are used. However in the ATL system, changing software models does not require re-compilation of the hardware processor model. The ATL library consists of a processor element, switch element and a shared memory element model. The processor element is conceptually divided into two concurrent processes: the computation agent and the communications agent. The computation agent has four basic instructions: compute, send, receive, and loop.

In general, the differences between the ATL and PML models are that the PML was designed to be more general purpose and configurable. The trade-off is that the PML models allow rapid model construction at some performance expense. The ATL models are more custom in nature but are somewhat faster. This is a classic trade-off in modeling systems.

RASSP Token-based Performance Modeling Application Note

5.0 Performance Modeling Method

5.1 Descriptive paradigm

The Processor-Memory-Switch (PMS) paradigm [9] describes the hardware architecture of a processing system as the structural interconnection of PEs, network switch elements (SEs), and shared memory elements (MEs). The interconnecting links are considered as monolithic data conduits that may represent, for example, fiber, coax, twisted pair, or bundles of conductors. The links are characterized by their data transfer rate, fixed transfer latency, and protocol relative to its performance as a function of demanded load. The PEs, SEs, and MEs are described behaviorally, and therefore possess no further decomposition of internal structure as is consistent with network performance modeling.

5.2 HW/SW-Codesign process

The hardware/software co-design process is characterized by making design decisions and trade-offs between hardware and software in a cooperative and iterative fashion. This concept is in contrast to traditional approaches where the hardware architecture is selected and set prior to designing the software, or vice versa. **Approaches without co-design preclude potentially superior solutions because decisions are made in the absence of total design information.**

In ATL's RASSP concept, co-design begins immediately after the requirements for the signal processing subsystem have been established and continues throughout the remainder of the process. It begins with trade-offs in the initial decisions and occurs at an abstract level. The co-design continues with joint trade-offs occurring down to the very detailed levels of hardware and software, where applicable.

Figure 5 - 1 below illustrates the co-design concept as applied to Lockheed Martin ATL's performance modeling environment. The shaded block on the left indicates the description of the candidate hardware system, while the shaded block on the right indicates the description of the system's application software. The hardware is described as the topological interconnection of the building-block element models. The software is described at several levels beginning with the data-flow-graph (DFG) of the application and resolved to sequences of abstract software tasks for each target processor element. The target software programs are the result of a partitioning, mapping, and scheduling process. The proposed hardware and software design are brought together during simulation. The system is simulated as the software executing on the hardware. The abstract target software programs are interpreted by the abstract hardware models. The results of simulation consist of time-lines and utilization statistics that are then analyzed for improvements to the hardware, the software, or combinations of both.

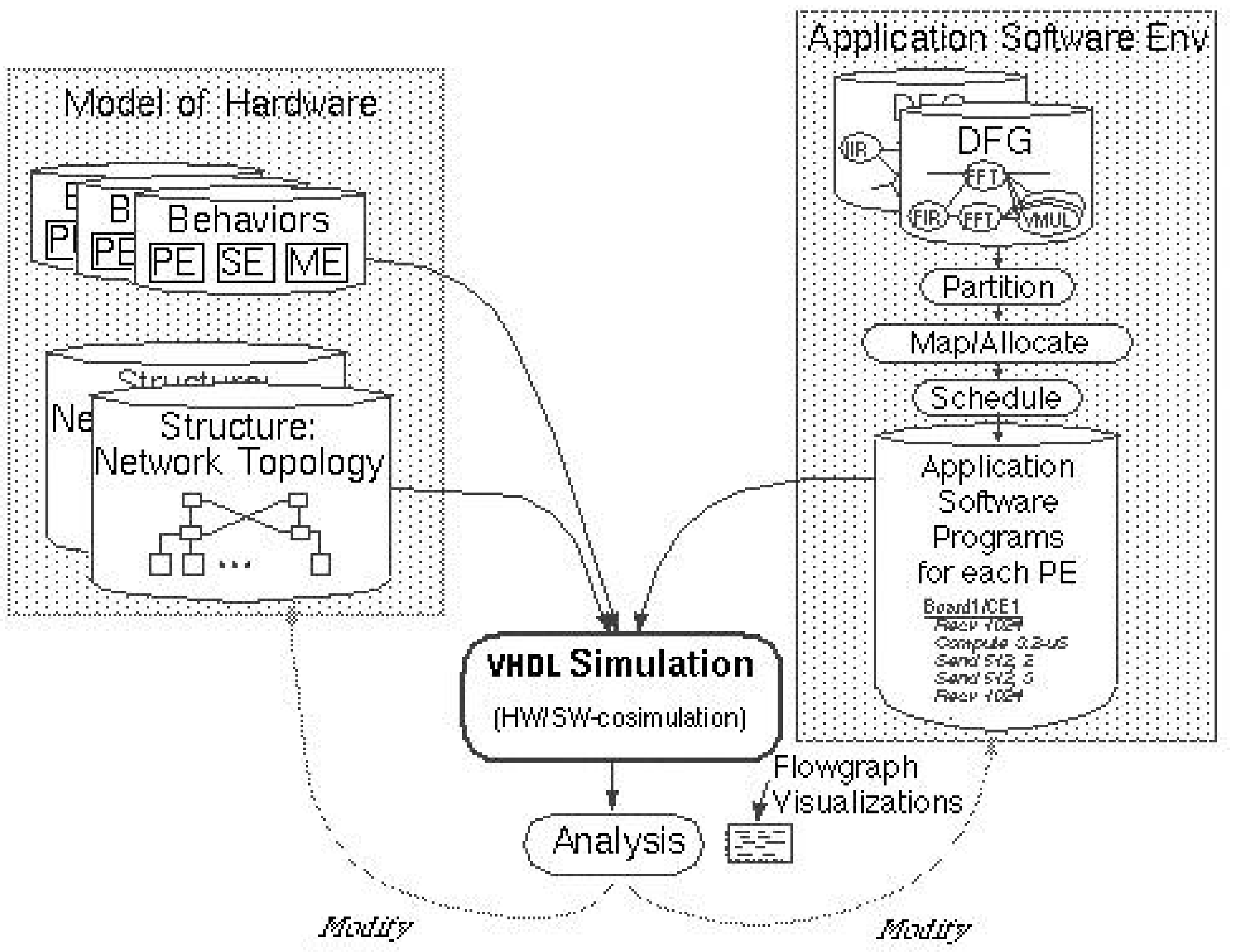


Figure 5 - 1: Hardware/Software Co-Design Process facilitated by performance simulation.

The following sections summarize the steps to be used for conducting performance modeling. This discussion is particularly appropriate to the ATL performance model library. However, similar methods apply for the Adept, Cosmos, or other modeling environments. The outlined methods can be adapted accordingly..

5.3 Steps for Token-Based Performance Modeling

The following sections summarize the steps to be used for conducting performance modeling. This discussion is particularly appropriate to the ATL performance model library. However, similar methods apply for the Adept, Cosmos, or other modeling environments. The outlined methods can be adapted accordingly.

5.3.1 Hardware Description

The following are guidelines for selecting the appropriate model abstraction level for network modeling. These are especially useful for multiple-instruction, multiple-data (MIMD) architectures with large granularity mappings.

Resolved events should be on the order of thousands of clock-cycles. For example, the begin and end events could be resolved for a data transfer as shown in figure 5 - 2, or for a PE computing a vector arithmetic operation, such as Fast-Fourier-Transform (FFT), as opposed to a single clock-cycle scalar operation.

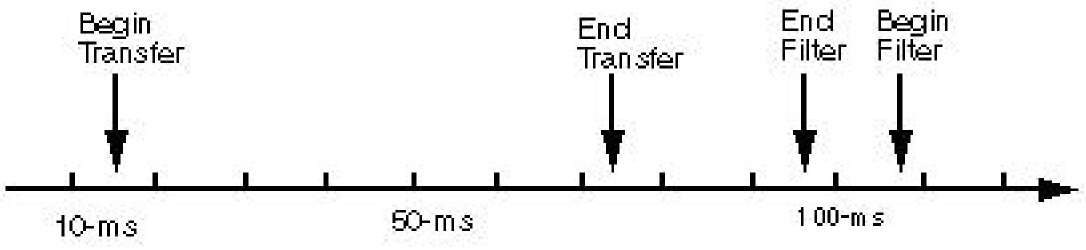


Figure 5 - 2: Resolution of time-events.

Contention for memory, communication, and computation resources should be resolved to accurately account for competing interactions. This can be done by allocating specific resources for the period of use and blocking competing operations while needed resources are not available.

Major system events of interest should be modeled for visibility into the processing. For instance, the transition between major sections of an algorithm can be identified.

In general, smaller sequential events whose time-delay and sequence can be accurately predicted should be aggregated into a single pair of begin-end events representing the start and conclusion of the group. The largest groups, for which accurate time delays can be predicted, should be formed. For example, the uni-processor tasks between inter-PE communication events of a partitioned algorithm can usually be aggregated.

Inter-device communication events should be resolved to account for network traffic. Communications should be resolved only down to the packet or message level as opposed to word transfer level. For instance, only the beginning and ending of a packet transfer need be resolved, assuming that the time for the packet transfer to complete (once started) is determined by the packet length, the transfer rate, and a fixed overhead.

Processor Element

The PE for a Multiple-Instruction-Multiple- Data (MIMD) system is conceptually divided into two concurrent processes: the computation agent, and the communications agent. The PE contains local memory for storage of the software program and working data as shown in figure 5 - 3. The performance model of the PE does not store any actual data. Rather, it keeps track of how much data would be stored in various logical queues by the application algorithm.

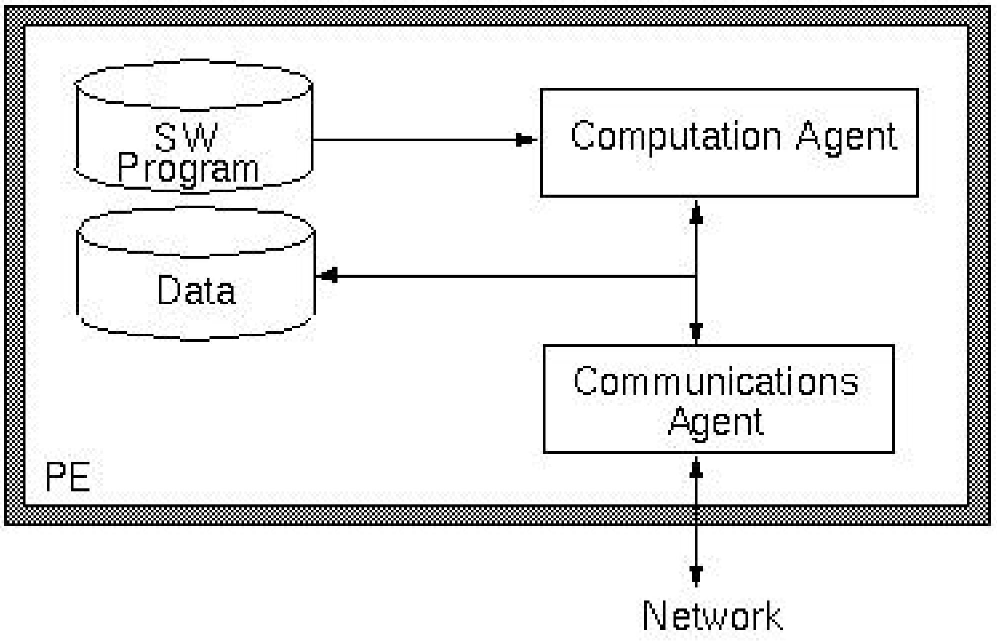


Figure 5 - 3: Processor Element Model

The communications agent handles the reliable transfer of data between the other PEs and the local PE's memory queues. It implements whatever link layer protocols, packetization, and retry or blocked message resumption that are needed to transfer and receive arbitrary length data messages over the network. Upon reception of data, the communications agent increments the data amount of the destination queue by the received amount. If the computation agent was blocked waiting for the received data, the communications agent would allow the computation agent to resume. Likewise, upon sending data, the communications agent decrements the data amount of the local source queue by the transmitted amount.

The computation agent represents the hardware side of the interface between the hardware and software because it interprets the software application program instructions into specific hardware actions. The computation agent executes a partitioned flow graph. A simple example of a computation agent for a statically scheduled, single-thread-per-PE system is described here. Extensions to other cases can be made as appropriate. Within the scope of the network performance level, the abstract instruction set of the computation agent may consist of four basic instructions: compute, send, receive, and loop. Although these instructions are abstract, their interpretation by the PE performance model is perfectly analogous to assembly code execution by an ISA model. The computation agent maintains a program counter to keep track of the software application program instruction it is executing.

The compute instruction represents the execution of a portion of the application algorithm within the PE's local memory. It is modeled in the performance model as a simple time delay. The compute instruction contains one operand specifying an algorithm step or corresponding computation time. The length of the time delay is equal to the time required for the target PE to perform the respective algorithm step. The time-delay value depends both on the type of PE and on the operations contained in that step of the algorithm. The time values can be obtained a variety of ways depending on the case. For COTS PEs, reliable time measurements for common processing functions, such as FFT, or vector multiply can often be obtained from data books and other published sources. Benchmark measurements of actual or typical algorithm segments can also be taken from an ISA simulation model or physical PE for a COTS processor when reliable measurements for the required operations cannot otherwise be obtained. For custom PE's that have not been constructed, either quick estimates based on intrinsic operation counts and the projected PE operation rate can be made or benchmarks can be taken from ISA simulation models. Upon completion of a computation delay interval, the computation agent interprets the next sequential instruction in the software application program. Because this is a performance model, no application computations are actually performed in the model.

The send instruction represents an inter-PE data transfer. It contains three operands: the local and destination queue numbers, and the data amount. Other operands, such as priority may be modeled. When the computation agent encounters a send instruction, the computation agent directs the local communication agent to transfer data from a local memory queue to a queue in another PE. If the communication agent can accept the command immediately, the computation agent continues sequencing to the next instruction in the software application program. Otherwise, the computation agent blocks execution until the communication agent resumes. Many systems feature a command queue for the communication agent that can be modeled to minimize such blocking. No data is actually transferred in a performance model. The model describes only the effects of transferring data, such as port allocation for the amount of time required to send the specified data amount and memory allocation amount for storing the equivalent data.

The receive instruction represents the consumption of transferred data. It has two operands: queue number and data amount. If the sufficient amount of data had arrived in the specified queue prior to encountering a receive command, then the computation agent decrements the specified queue by the specified receive amount. Then it continues to sequence to the next instruction in the software application program. Otherwise, the computation agent blocks until sufficient data in the specified queue arrives.

Switch Element (SE)

The SEs are multiple port entities that route data packets or messages from one port to another port. When connected to other SEs via links to form a network, the SEs provide a means to transfer data from one PE to any other PE or ME within the processing system. For a network-performance model, no data is actually transferred over the links. However, a link's bandwidth is allocated for the appropriate duration of a data transfer to account for the movement of data over the given link. Various switching schemes may be modeled, such as common-bus, circuit-switched, packet-switched, or store-and-forward. Each scheme exhibits unique behavior under contention for common network links by competing PE nodes. The effects of such contentions are especially critical to the successful design of real-time, high-throughput, signal processors for many applications. An SE can be modeled as a set of processes handling the activity at each of the ports, as shown in figure 5 - 4.

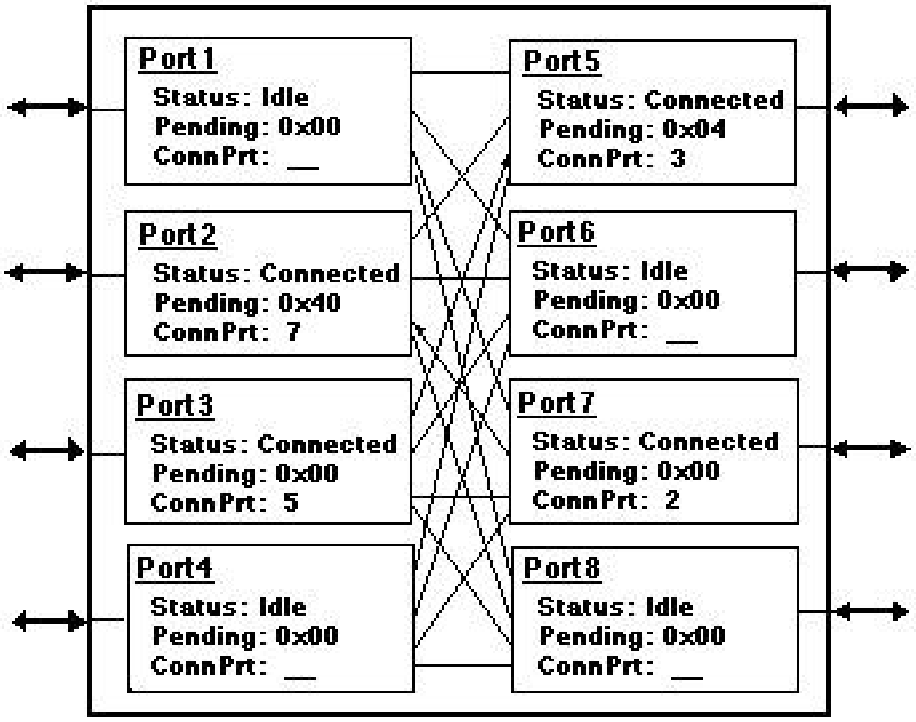


Figure 5 - 4: Switch Element Model for Crossbar

Shared Memory Element (ME)

The shared ME represents a common data storage resource accessible by PEs over the network. Its model and role are similar to that of the local memory of a PE.

Modeling Issues and Techniques

The PMS paradigm described above can be implemented with VHDL in various ways. We advocate a direct approach where the network topology is described directly in terms of a VHDL structural description. Because the physical structure of digital systems typically consists of a hierarchy of modules, boards, chassis, and racks, we pattern the structural hierarchy after the physical hierarchy. The PEs, SEs, and MEs become the leaf-level components of the structural description. The signal links of the structural models interconnect the leaf-level components to each other.

Because the abstract network-level paradigm transfers only symbolic tokens representing data messages instead of actual data values, a token composite type must be defined. The signals and component ports are declared to be of type token.

The use of a common token definition is critical for the re-use and interoperability of abstract models from diverse sources such as libraries and other project groups. Honeywell Technology Center has proposed a token type convention for performance modeling [10], as shown below.

```
TYPE utoken IS
 RECORD
  destination   : name_type;
  source        : name_type;
  t_type        : token_type;
  size          : data_size;
  value         : INTEGER;
  id            : uGIDType;
  start_time    : TIME;
  priority      : INTEGER;
  state         : State_type;
  protocol      : Protocol_Type;
  collisions    : INTEGER;
  retries       : INTEGER;
  route         : INTEGER;
  param_int     : INTEGER;
 END_RECORD
```

The behaviors of the network components (PE, ME and SE) are modeled in procedural VHDL in accordance with the paradigm described in section 5.1, 5.3.1 and 5.3.2. Because the duration of modeled events in on the order of thousands of clock cycles, the models should be asynchronous, event-driven models, as opposed to synchronous clock-driven models. This minimizes the number of events to be executed by the VHDL simulator and avoids the inefficiency of evaluating many clock events for which no meaningful system event occurs.

5.3.2 Software Description

In the ATL approach, the signal processing application algorithm is first represented as a data flow graph (DFG). The DFG is a directed graph that describes an application algorithm in terms of the inherent data dependencies of its mathematical operations. The graph nodes represent mathematical operations, and the arcs that interconnect the nodes represent the data dependencies and form the logical data queues. The DFG conveys the potential concurrencies within an algorithm, which facilitates parallelization and mapping to arbitrary architectures. The DFG nodes usually correspond to DSP primitives, such as FFT, vector multiply, convolve, or correlate.

For a given network architecture, the application flow graph is partitioned for allocation to PEs in the system.

Shared Memory Element (ME)

The shared ME represents a common data storage resource accessible by PEs over the network. Its model and role are similar to that of the local memory of a PE.

Modeling Issues and Techniques

The PMS paradigm described above can be implemented with VHDL in various ways. We advocate a direct approach where the network topology is described directly in terms of a VHDL structural description. Because the physical structure of digital systems typically consists of a hierarchy of modules, boards, chassis, and racks, we pattern the structural hierarchy after the physical hierarchy. The PEs, SEs, and MEs become the leaf-level components of the structural description. The signal links of the structural models interconnect the leaf-level components to each other.

Because the abstract network-level paradigm transfers only symbolic tokens representing data messages instead of actual data values, a token composite type must be defined. The signals and component ports are declared to be of type token.

The use of a common token definition is critical for the re-use and interoperability of abstract models from diverse sources such as libraries and other project groups. Honeywell Technology Center has proposed a token type convention for performance modeling [10], as shown below.

```
TYPE utoken IS
 RECORD
  destination   : name_type;
  source        : name_type;
  t_type        : token_type;
  size          : data_size;
  value         : INTEGER;
  id            : uGIDType;
  start_time    : TIME;
  priority      : INTEGER;
  state         : State_type;
  protocol      : Protocol_Type;
  collisions    : INTEGER;
  retries       : INTEGER;
  route         : INTEGER;
  param_int     : INTEGER;
 END_RECORD
```

The behaviors of the network components (PE, ME and SE) are modeled in procedural VHDL in accordance with the paradigm described in section 5.1, 5.3.1 and 5.3.2. Because the duration of modeled events in on the order of thousands of clock cycles, the models should be asynchronous, event-driven models, as opposed to synchronous clock-driven models. This minimizes the number of events to be executed by the VHDL simulator and avoids the inefficiency of evaluating many clock events for which no meaningful system event occurs.

5.3.2 Software Description

In the ATL approach, the signal processing application algorithm is first represented as a data flow graph (DFG). The DFG is a directed graph that describes an application algorithm in terms of the inherent data dependencies of its mathematical operations. The graph nodes represent mathematical operations, and the arcs that interconnect the nodes represent the data dependencies and form the logical data queues. The DFG conveys the potential concurrencies within an algorithm, which facilitates parallelization and mapping to arbitrary architectures. The DFG nodes usually correspond to DSP primitives, such as FFT, vector multiply, convolve, or correlate.

For a given network architecture, the application flow graph is partitioned for allocation to PEs in the system.

The partitioned flow graph nodes may be allocated statically at design-time or dynamically at run-time. In either case, the tasks may be scheduled for execution statically or dynamically. The subject of partitioning/mapping/scheduling remains an open research topic that is beyond the scope of this discussion [11,12,13]. However, the paradigm described here allows either of the cases to be modeled. Dynamic allocation and scheduling requires modeling the dynamic mapper and scheduler. Static allocation and scheduling requires the mapping and scheduling to be done prior to simulation. The regularity of many DSP applications allows static scheduling, as described in this paper.

The static partitioning/mapping/scheduling process produces a set of pseudo-code software application programs for each of the PEs. The scheduling determines only the order of tasks executed by a given PE. The actual time when execution begins for each task is determined by the task sequence and the inherent data flow control of the send/receive paradigm. The PE programs are expressed as a sequence of pseudo-code instructions from the simple instruction set described in section 5.3.1 under Processor Element. New mappings and schedules can be tested by rearranging the instructions accordingly.

Once simulations show a suitable software mapping and hardware architecture combination to satisfy the system performance requirements, the pseudo-code software routines are expanded into high-level-language subroutine calls, which are compiled for down-loading to the target hardware or more detailed ISA models for verification of the constituent performance factors. The send/receive calls are substituted with the appropriate communication routines for the target system. The compute instructions are substituted by calls to the appropriate DSP library routines or functions.

5.3.3 Simulation

At the start of simulation, the hardware models read their respective application software programs and begin to interpret them. The interpretation of the software programs causes the processor elements to send and receive messages over the network, and to delay for specific computation events. The designer can set break-points for specific times to examine the simulated system's status. VHDL simulators provide extensive capabilities for viewing the values of each model's internal states. This is very helpful while debugging. During simulation, event-history information is recorded into files for post-processing analysis.

5.3.4 Postprocessing/Analysis

To visualize the result of a simulation, the event-history file can be translated into an xy-graph format for plotting the time-line information. A useful event format is as follows:

```
    device @ time:  event-string
```

Where *device* is the name of the entity on which the event occurred. *Time* is the time at which the event occurred. and the *event-string* is a meaningful description or name of the event. For example:

```
    /board1/PE_03 @ 1923.084:  Began FFT_1024
    /board4/xbar7 @ 1925.921:  Transferred packet
```

5.3.5 Recursion

To determine the sensitivity of a design parameter, it is often useful to execute a simulation iteratively: each time changing the parameter slightly. It is usually convenient to set up the simulation recursion information in a script file so that the recursions are run automatically. The information can be collected and displayed automatically as well.

5.3.6 Model Validation/Maintenance

As the design process progresses, a performance model's accuracy should be continually checked against more detailed models as they become available or to measurements from the actual components. Any mismatch should be corrected to maintain the performance model's accuracy, to test for continued compliance with requirements, and to support subsequent re-use and model-year upgrades. This activity departs from traditional processes which do not maintain -and therefore effectively discard- the performance model once the architecture design has completed.

Examples of model aspects that can be incrementally refined include the network loading behavior and the task primitive execution times which may have been initially based on estimates.

For instance, accurate timing values can be obtained from an application code segment running on an Instruction Set Architecture (ISA) model of a target processor element (PE). The new values update the task execution time lookup-table for the given PE type. For example, on the RASSP Benchmark - II Synthetic Aperture Radar (SAR) design project [14], the values of the performance model's task execution time table were updated with measurements from a physical development system.

Initially, the SAR task partitioning was determined and validated by performance simulations based on estimates from a summation of the published execution times of the individual vector functions that comprised the various application tasks. Then each of the aggregate tasks was executed on a single target PE that was available on a development board. The actual execution times were compared against the estimates to check for consistency and then the actual time values were used in re-simulating the full system running the complete application to assure that the appropriate design margins were retained or to re-partition if needed.

Another instance of model refinement from the SAR design project involved the resolution of network protocol. The initial models resolved the transfer of data between PE's down only to the message level. However, intercommunication benchmarks showed that under moderate to heavy traffic loads the performance predicted by the modeled system deviated substantially from that observed on a small physical development system. The inconsistency was traced to the effects of contention and the packetization of messages into finite length packets on the target system. It was found that by resolving the message packetization process, the model's behavior was brought into consistency with the observed performance. The model was validated as a result of this process.

Next Up Previous Contents

Next: 6 Example: SAR System Application **Up:** Appnotes Index **Previous:**4 Performance Modeling Environments

RASSP Token-based Performance Modeling Application Note

6.0 Example: SAR System Application

The Digital Signal Processing (DSP) subsystem of a Synthetic Aperture Radar (SAR) system was modeled under the RASSP Benchmark - 2 project. The performance modeling was conducted by Lockheed Martin's Advanced Technology Laboratories (ATL) in VHDL for relatively seamless transition to the down-stream detailed design processes. The initial design risk assessment indicated the significant challenges involved integrating and coordinating the various hardware and software elements into a system of multiple cooperating PE's. In particular, the application's real-time operation presented the highest risk. Because the hardware and firmware elements were selected from COTS products, they effectively became validated by default. Therefore, the design team considered the simulation of the complete hardware-software multi-processor system to be more important than simulating the operation of an individual sub-section of the architecture.

Software Description

The data flow graph (DFG) of the SAR application is pictured in figure 6- 1 below.

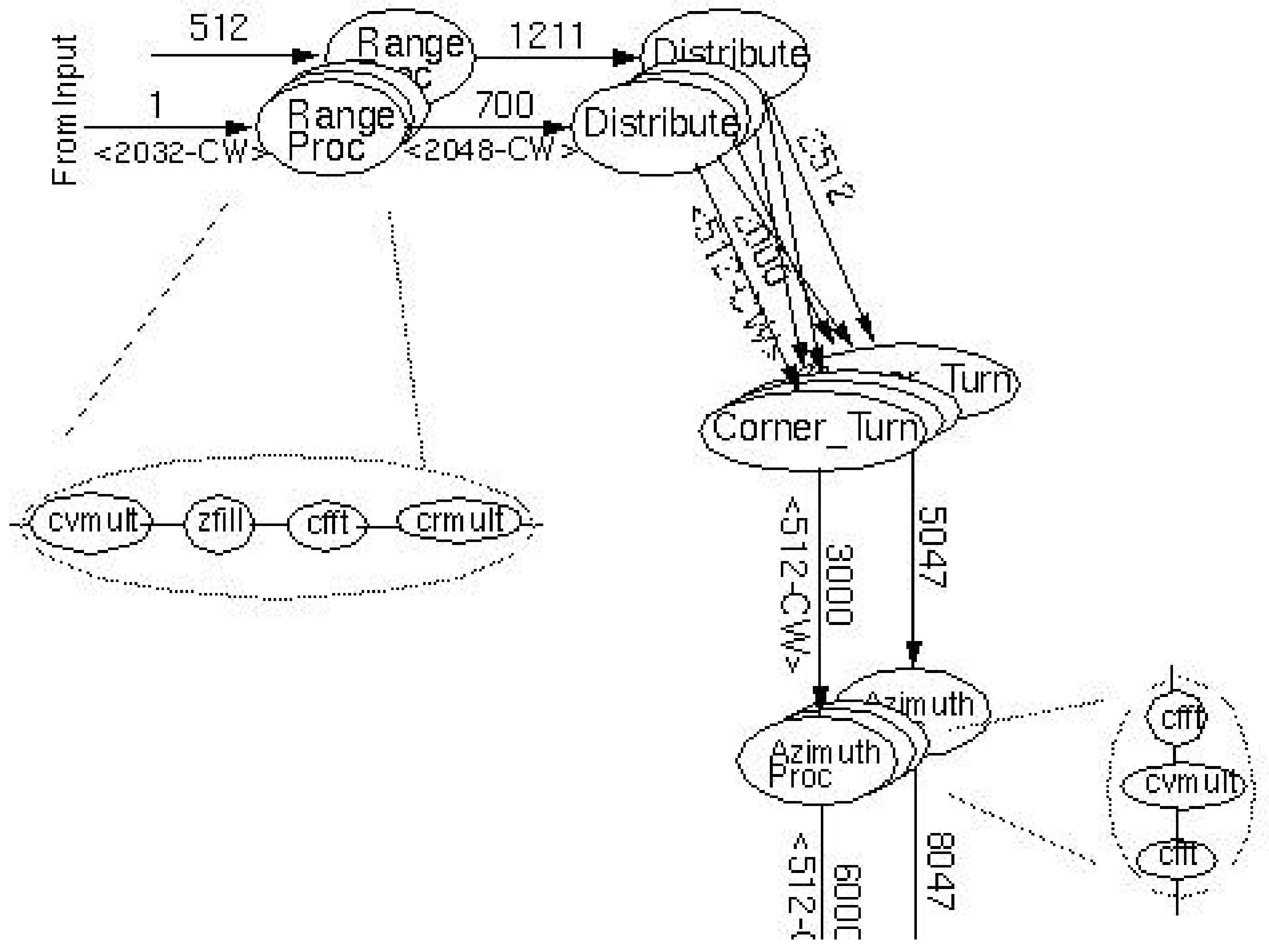


Figure 6 - 1: SAR Application Data Flow Graph.

The application DFG contains about 11,000 task nodes. The task nodes were assigned to processor elements (PEs) by partitioning the DFG into tightly-connected groups of nodes.

Hardware Description

The processing system consisted of 24 Processor Elements (PEs) and multiple crossbar elements. Two candidate architectures were evaluated with the token-based performance models. The first was the Mercury-Raceway(TM) based network, figure 6- 2.

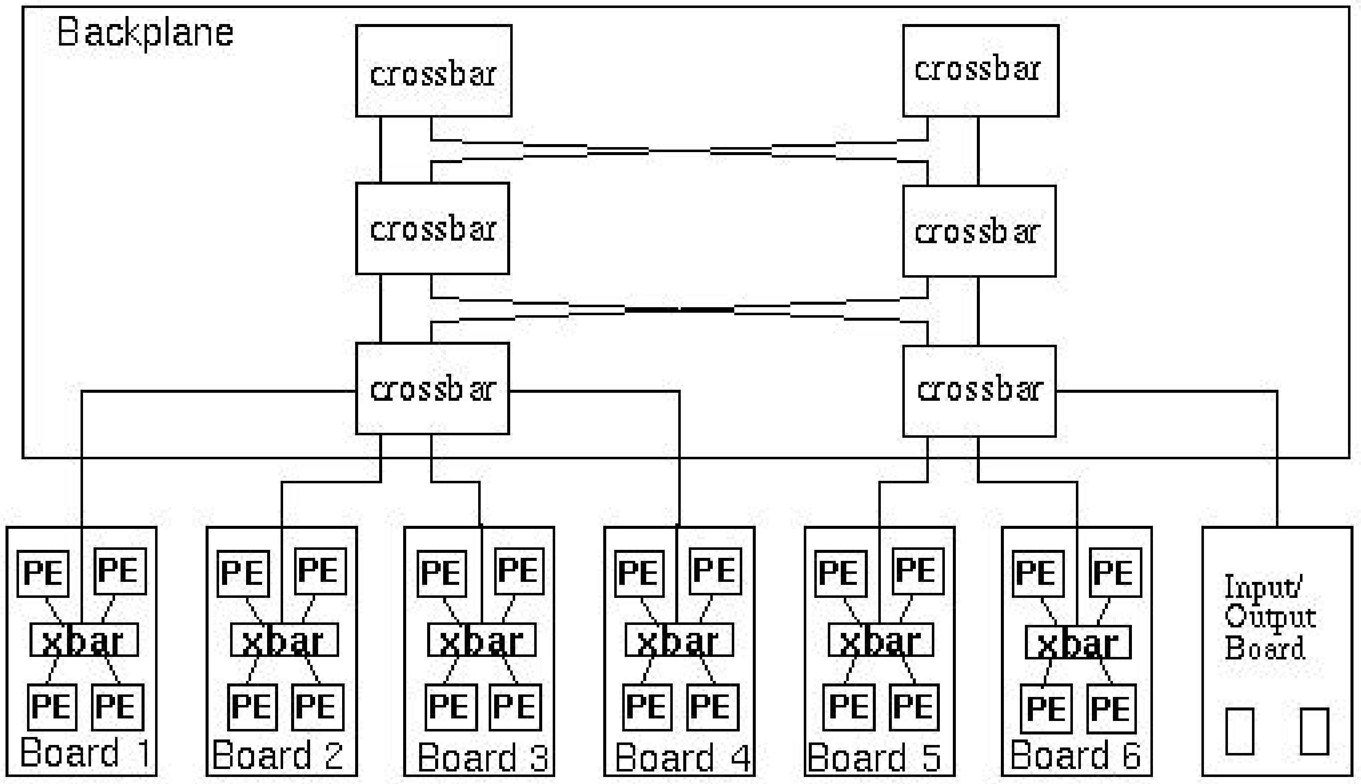
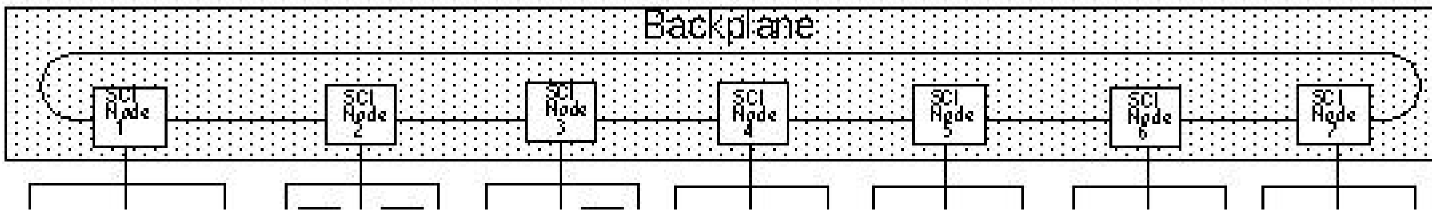


Figure 6 - 2: ATL SAR Architecture Candidate 1: Mercury Computer Raceway based network

The second candidate architecture was a Scalable Coherent Interface (SCI) based network pictured in figure 6-3. The Raceway network belongs to the multi-stage switch network architecture class, while the SCI is a ring network type.



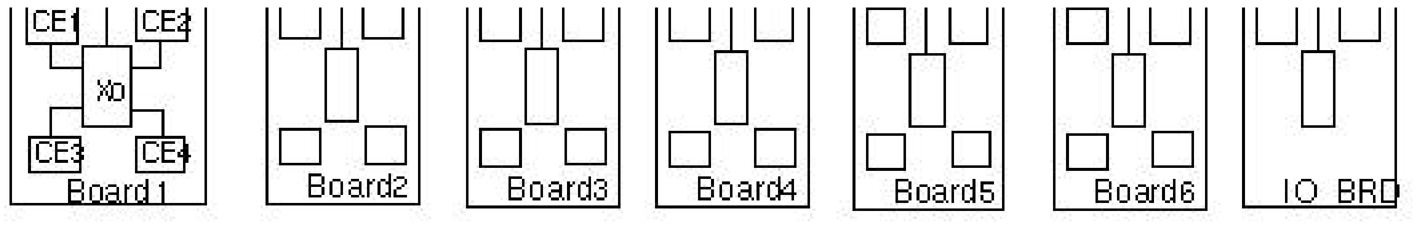


Figure 6 - 3: ATL SAR Architecture Candidate 2: Scalable Coherent Interface (SCI) based network.

Rapidly simulating a significant portion (5 - seconds) of the real-time application executing on the full multi-processor system required a much higher efficiency and modeling abstraction than that of the typical ISA-level model. To be abstract, yet accurate, only the necessary details were resolved in the model. These included significant protocol events such as initiation and completion of data transfers as well as significant computational events such as the beginnings and endings of bounded computational tasks. The resolved events focus around contention for computation and communication resources whose usage time, once allocated for a task or transfer, was highly deterministic. The simulation is valuable because the contention for the multiple resources is not conveniently predictable.

The nature of the SAR application implied a relatively large computational event granularity; on the order of thousands of clock-cycles. Similarly the RASSP design group found that resolving the communication events to the packet level accurately characterized the communication network performance.

The design the models of the Raceway communication network, began with a description of the communication protocol in a way that was as abstract as possible while still providing an accurate depiction of its resultant performance. The protocol was described by its significant events, which were the beginning and ending of packets.

To implement the protocols in VHDL, the abstract protocols were expressed as state transition diagrams. The state transition diagrams are conveniently converted to VHDL by means of a case statement on the state.

For more specific information on the VHDL code details, with excerpts of actual VHDL code, see VHDL detailed excerpts.

Results

The design group produced time-line graphs from the simulation results which showed the history of task executions on the PE's. The graphs were useful in helping to visualize and understand the impact of mapping options that led the design group to modify, optimize, and ultimately verify the partitioning, allocation, and scheduling of the software tasks onto the hardware elements. The time-line graphs showed the times when PE's were idle due to data starvation or buffer saturation that helped isolate other resource contentions and bottle-necks.

The processing time-line graph in figure 6 - 4 is an example of the kind of visual output obtained from token-based performance modeling.

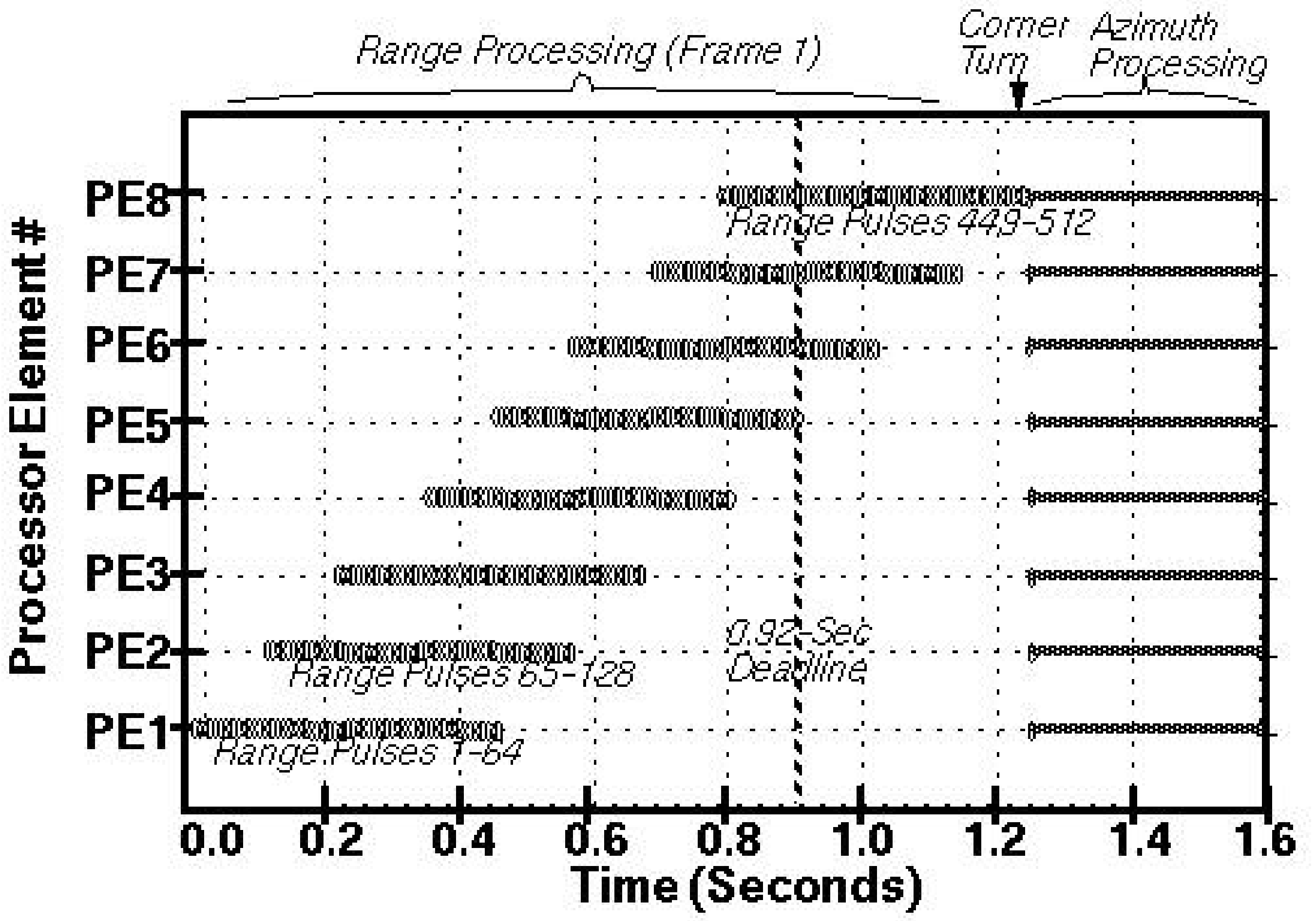


Figure 6 - 4: Example Time-Line Graph (sub-optimal).

To illustrate the usefulness of token-based performance modeling, the first graph shows a sub-optimal mapping of tasks to processors. Notice that there is much idleness of the processors on the left half of the graph. This mapping resulted in poor performance, as the processing deadlines were not met. Visualizing that initial time-line led to an alternative mapping strategy that allocated processing tasks in a finer-grain round-robin style. The resulting processing time-line graph is shown in figure 6 - 5 below.

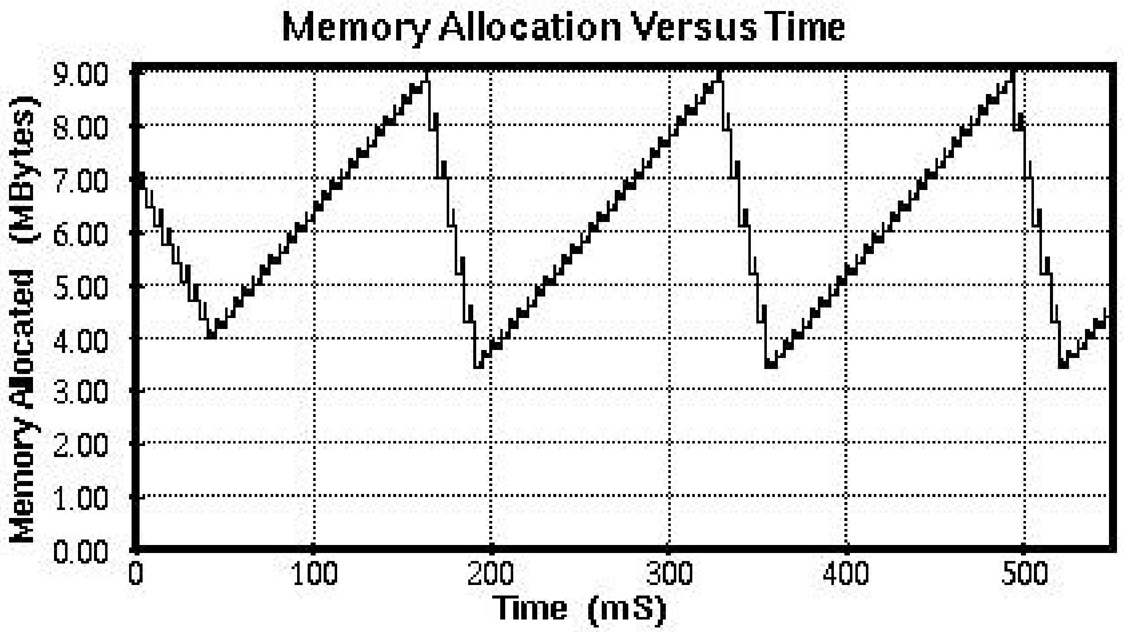


Figure 6 - 6: Example Memory Allocation Graph.

Efficiency

The VHDL performance simulation consumed 28 Sparc - 10 CPU minutes. This implies an equivalent instruction execution rate of about 2.9 - million per second when considering the number of PEs and their individual instruction rates. The corresponding abstract-behavioral model consumed 14 CPU-hours and exhibited an effective execution rate of 23,810 instructions per second.

By comparison, a much more concrete model, such as an ISA-level VHDL model of the i860 PE, exhibited about 5.5 to 7.5 instructions-per-second on a Sparc - 10 CPU [6]. Such ISA models are more useful for understanding the behavior of software segments that dwell within a given PE.

Simulation	Simulated Time	Host CPU Time	Equiv. Instructions/Sec
VHDL Performance Model of 24-PE System *	5.0-Sec	28-Minutes	2,857,000
VHDL Performance Model of 18-PE System **	5.0-Sec	18-Minutes	3,333,000
VHDL Abstract Behavior Model of 6-PE System **	5.0-Sec	14-Hours	23,810
ISA Model of One i860 PE Node *	5.0-mS	12-Hours	5

*i860-XR 40-MHz, 40-MIPS
** ADSP-21060 Sharc 40-MHz, 40-MIPS

Table 6 - 1: Comparison of simulation efficiencies for types of models in the design process.

Down-Stream Process

The resultant software task partitioning and schedules from the performance model led directly to the production of the target source code through a straight-forward translation into sub-routine calls.

Accuracy

When the physical SAR DSP system was built and loaded with the generated application software developed in the modeling process, the design group found that their simulations accurately predicted the system's actual run-time performance to within a few percent. The DSP system's processing throughput requirements were satisfied without further modification.

The time required to develop the VHDL Token-based performance model of the system was about 5 - weeks with two engineers. The total time consumed in the development activity was 371 - hours. About 1378 Source-Lines-Of-Code (SLOC) were generated for the models. Additionally, about 1657-SLOC were generated for the test-benches for verifying the correctness of the models and to assist in their development.

Future efforts should require much less time, as the original effort included significant learning-curve time for developing methods for describing performance models in VHDL as well as the time to develop all models from scratch. Subsequent projects are reusing the previously developed models with greatly reduced design time.

Model Library

The full versions of the actual VHDL models are obtainable from the following repository: ATL Performance Model Library.

Next Up Previous Contents
Next: 7 References **Up:** Appnotes Index **Previous:**5 Performance Modeling Method

RASSP Token-based Performance Modeling Application Note

References

IEEE Standard VHDL Language Reference Manual, IEEE Std 1076 - 1993, IEEE Customer Service, 445 Hoes Lane, PO Box 1331, Piscataway, New Jersey 08855 - 1331.[Purchase at the above address]

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. [Purchase from Kluwer Academic Publishers]

Holliday, M. A., M. K. Vernon, "A Generalized Timed Petri Net for Performance Analysis," IEEE Transactions on Software Engineering, Vol. SE - 13, No. 12, December 1987.[Reference Not Available]

Chang, C. K., Y. Chang, L. Yang, C. Chou, J. Chen, "Modeling a Real - Time Multitasking System in a Timed PQ Net," IEEE Software, March 1989, pp. 46 - 51. Ferrari, D., Computer Systems Performance Evaluation, Prentice-Hall, Englewood Cliffs, NJ, 1978. [Reference Not Available]

Rose, F., T. Steeves, and T. Carpenter, "VHDL Performance Models," Proceedings 1st Annual RASSP Conference, pp 60 - 70, Arlington, VA, August, 1994. [ROSE_94]

Steeves, T., et al, "Evaluating Distributed Multiprocessor Designs," Proceedings 2nd Annual RASSP Conference, pp 95 - 102, Arlington, VA, July, 1995. [STEEVES_95]

Shackleton, J., T.Steeves, "Advanced Multiprocessor System Modeling," Proceedings Fall 1995 VIUF, pp 8.21 - 8.28, Boston, MA, October, 1995 [SHACKLETON_95]

Hein, C., and D. Nasoff, "VHDL - based Performance Modeling and Virtual Prototyping", Proceedings 2nd Annual RASSP Conference, pp 87 - 94, Arlington, VA, July, 1995. [HEIN_95]

Siewwork, Newell, and Bell, "Computer Structures Principles and Examples", MCGraw-Hill, 1982. [Purchase from McGraw-Hill publishers]

Honeywell Technology Center, "VHDL Performance Modeling Interoperability Guideline", available on-line at: PMIG and related docs. [HONEYWELL]

Jain, R., Werth, J., "A General Model for Scheduling of Parallel Computations and its Application to Parallel I/O Operations", Proceedings of the 1991 International Conference on Parallel Processing, April, 1991. [Reference Not Available]

Yang, J., Bic, L., "A Mapping Strategy for MIMD Computers", Proceedings of the 1991 International Conference on Parallel Processing, April, 1991. [Reference Not Available]

Agrawal, D., Shukla, S., "Allocation and Communication in Distributed Memory Multiprocessors for Periodic Real-Time Applications", Proceedings of the 1991 International Conference on Parallel Processing, April, 1991. [Reference Not Available]

Zuerndorfer, B., Shaw, G., "SAR Processing for RASSP Application", Proceedings of the RASSP

Conference, August, 1994. [ZUERNDORFER_94]

7.1 Application Notes

Hardware/Software Codesign
System Process
VHDL Taxonomy
Virtual Prototyping

Next Up Previous Contents

Next: Up: Appnotes Index **Previous:**6 Example: SAR System Application