VHDL-Based Performance Modeling and Virtual Prototyping

Carl Hein and David Nasoff
Lockheed Martin
Advanced Technology Laboratories
Camden, NJ 08102
chein@atl.ge.com dnasoff@atl.ge.com

Abstract

The need for top-down, hierarchical, simulation processes in RASSP has pushed the use of VHDL at levels of abstraction above where it has typically been used. Selection of an appropriate modeling abstraction level permits rapid exploration of many alternative software and hardware solutions while achieving accurate yet efficient simulation. Performance and virtual prototype models facilitate the integrated design and development of the software and the hardware subsystems in a hardware/software codesign process. In this paper, the Lockheed Martin Advnced Technology Laboratories (ATL) RASSP team will present for producing an efficient VHDL-based performance model of a Digital Signal Processor (DSP) system, which is then extended to form a comprehensive virtual prototype of a full multi-processor DSP system that is both timing- and data-faithful. The system models provide early design verification via simulation of software, as partitioned, mapped, and executed on the hardware architecture. This paper will focus on VHDL-specific issues and techniques for this type of very abstract modeling.

1 Introduction

Digital Signal Processing (DSP) systems become ever more complex with increasing technology and integration levels. Their efficient design requires initiating the design process at an abstract level and relying on a common language to transfer design information between every stage of the process from top to bottom as the abstractions are resolved into details. VHDL is uniquely capable of expressing the temporal, structural, and functional relationships at the required abstraction levels.

VHDL is commonly used at the logic, register transfer level (RTL) and chip behavioral level, while special-purpose languages have been used at the DSP-system and network-architecture levels. Network-based multiprocessing is a convenient way to expand processing power for a given state of processor technology. This paper focuses on methods for applying VHDL for DSP-system and network-architecture design in a mode that is extensible for supporting and interacting with the subsequent design levels.

Comprehensive simulation of the logic circuits on and between individual integrated circuits (ICs) has reduced the number of design errors in the internal logic of ICs, such that the majority of design faults now occur at the requirements-specification level for these devices. Consequently, it becomes expedient to focus on the system verification process, where the IC requirements are developed.

A performance model describes the time-related aspects of the system associated with response time, throughput, and utilization. Neither the actual application data nor the transforms on it are described other than that which are required to control the sequence of events. Virtual prototype models then describe the system with full timing, data, and functionality.

The approach presented in this paper allows verification of the full system concept prior to investing in the construction of an entire system. This approach is especially beneficial prior to designing or realizing custom processor elements (PEs), as well as in the case of commercial-off-the-shelf (COTS)-based systems, where only a one or a few PEs can be practically obtained for evaluation.

1.1 Purpose of Network-Performance Modeling

Network-performance modeling is performed to help in the determination of the system size, network architecture, software-to-hardware mapping, and performance requirements for each of the major constituent components of the DSP system. The ultimate processing latency, throughput, and physical constraints on the DSP system drive the optimization of the DSP architecture design.

The system size is determined in terms of the number and type of processing elements, and the memory and buffer requirements for each of the network and processor elements. For instance, specific mixtures of specialized PE types are optimally determined.

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 DSP systems include: bus, ring, mesh, cube, tree, and custom configurations. Performance simulations provide information concerning link and processor uses for specific topology, PE, and software mapping combinations.

The software-to-hardware mapping partitions the application algorithm into 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 instance, depending on the application, the mapping and scheduling can be assigned statically at design-time or dynamically at run-time.

1.2 Appropriate Abstraction Level

Unlike the testing of an IC's interface or functional design- which typically requires on the order of thousands of clock cycles or milli-seconds of simulated time- DSP-system simulation typically requires the simulation of significant portions of a DSP algorithm spanning several seconds of simulated time. The simulation of multiple cooperating PEs, each executing tens to thousands of millions of instruction cycles per second (MIPS), for several seconds represents the execution of an extraordinarily large number of events. Simulation of a multi-processor DSP system at or below a chip-behavior level, is therefore often impractical due to the large memory requirements and simulation run-times required.

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 potentially misguided detailed designs. The number of events must be reduced through abstraction without jeopardizing the accuracy or validity of the model. Therefore, selecting an appropriate modeling abstraction level is crucial for timely design. Note that abstraction does not imply a degradation in accuracy.

The following guidelines should be considered for selecting the appropriate model abstraction level for network modeling, especially for the currently popular multiple-instruction, multiple-data (MIMD) DSP 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 PE computing a vector arithmetic operation, such as Fast-Fourier-Transform (FFT), as opposed to a single clock-cycle scalar operation.

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.

1.3 Performance Modeling Requirements

In selecting a means to fulfill the purposes of network-performance modeling, several capabilities must be present. Assuming that a VHDL model describes the design of a hardware entity, then VHDL compilation is analogous to hardware fabrication. When a hardware design is modified, the VHDL model must be re-compiled just as the hardware must be re-fabricated. Conversely, when application software is changed, neither the processor chip or its VHDL model need to be re-fabricated nor analogously re-compiled, respectively.

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 paired to execute a variety of distinct application programs without modification to the hardware model other than directing it to access different software files. Conversely, a given software application program should be executable on a variety of candidate hardware configurations.

The network topology should be specified independently from the models of the network hardware components. Conversely, the component models should be modular so that network components, such as processor elements, can be interchanged without redesigning the network.

An important requirement 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 virtual prototypes.

Another important requirement is for *efficient models* that support quick simulation to permit the rapid exploration of many design alternatives.

2. Network Performance Paradigm

Consistent with the modeling requirements explained in section 1.3, we have adopted a scheme that captures congruent descriptions of the hardware and software aspects of a system. The respective model hierarchies are described below, beginning with the hardware architecture and followed by the complementary software hierarchy.

Although the following discussion centers around network performance modeling in which many functional and structural details are unresolved, the basic model framework persists and remains consistent throughout each stage of the ensuing design process. The models are simply expanded to add the appropriate details as the process descends toward synthesizable constructs and objects able to be down-loaded.

Because the partitioning, mapping, and scheduling problem is not-polynomial-(NP)-complete, there is no efficient, general, closed-form, optimum solution. Instead, many iterative, heuristic, and manual techniques are currently available. The models described in this paper facilitate many of these methods. Figure 1 shows the interactive development and optimization of a hardware and software design within an iterative mapping/scheduling process. The hardware/software codesign process begins with a model of the hardware onto which the software application tasks are mapped and scheduled. The joint hardware/software system is simulated, and modifications are made to either the hardware, software, or both in response to the analysis of the simulation results.

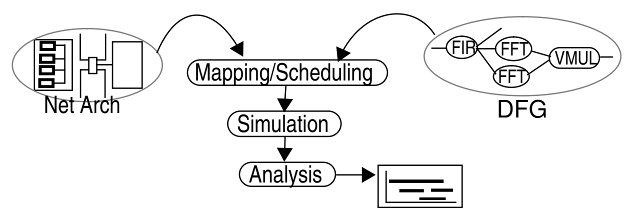

Figure 1 - Hardware/Software Codesign Process.

2.1 Hardware Architecture Model

To describe the hardware architecture of a DSP system, we selected the Processor-Memory-Switch (PMS) paradigm. The hardware architecture of the DSP system is described as the structural interconnection of processing elements (PEs), network switch elements (SEs), and shared memory elements (MEs), as shown in the example of figure 2. 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 their effective degradation 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.

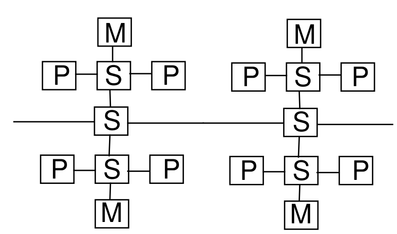

Figure 2 - Example DSP Architecture.

2.1.1 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 also contains local memory for storage of the software program and working data. 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.

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 cyclo-statically scheduled, single-thread-per-PE system is described here. Extensions to other cases can be made as appropriate. Within this scope for purposes at 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 DSP 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 on 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. The 1ci instruction 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. The *receive* command 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.

The *loop* instruction causes the computation agent to continue sequencing from a non-sequential instruction of the application program. The model accomplishes this by putting a new value into the program counter of the PE.

2.1.2 Switch Element

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 DSP 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.

2.1.3 Shared Memory Element

The 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.

2.2 Software Hierarchy

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. Figure 3 shows an example DFG.

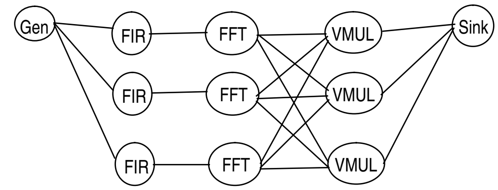

Figure 3 - Example of a Data Flow Graph.

For a given network architecture, the application flow graph is partitioned for allocation to PEs in the network. 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. However, the paradigm described here allows either of the cases to be modeled. Dynamic allocation/scheduling requires modeling the dynamic mapper/scheduler. Static allocation/ scheduling requires the mapping/scheduling to be done prior to simulation. The regularity of many DSP applications allows static scheduling, as described below.

The result of a static partitioning/mapping/ scheduling process is 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 2.1.1. 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.

3 VHDL 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 DSP systems typically consists of a hierarchy of modules, boards, chassis, and racks, we pattern the VHDL structural hierarchy after the physical hierarchy. The PEs, SEs, and MEs become the leaf-level components of the VHDL structural description. The signal links of the structural models interconnect the leaf-level components to each other.

Because our abstract network-level paradigm transfers only symbolic tokens, which represent data messages, over the VHDL signals 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 interoperability of abstract models from diverse sources such as libraries and other project groups. Under the RASSP subcontract to ATL, Honeywell Technology Center has proposed a token type convention for performance modeling [1], 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 are modeled in procedural VHDL in accordance with the paradigm described in section 2. Because the duration of modeled events is 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.

For efficiency, it is important to minimize the number of events used to model the network protocol. For instance, although the actual protocol being modeled may consist of four distinct events for transferring a packet of data, such as

1. sending a request (REQ),
2. receiving an acknowledge (ACK),
3. sending the data (DAT), and
4. lowering the request (NREQ).

The number of VHDL events can be reduced in half, without sacrificing accuracy in accounting for the use of the link. This can be accomplished by abstracting the transfer process to only the first and fourth event. Such an abstraction applies when the time of events two through four can be predicted deterministically from the time of the first event and any independent values such as the size of the data message and the link transfer rate.

If the protocol allows alternative events to follow the REQ, such as a negative-acknowledge (NACK) instead of an ACK, then the second event would be either an NREQ or NACK.

Although the specific behavior of VHDL signals simplifies the modeling of electrical conductors at the detailed levels, special techniques must be employed to circumvent these behaviors for more appropriate behaviors when modeling at higher levels of abstraction.

In the modeling of a distributed system with multi-port SEs, multiple transactions on a given port of an SE may sometimes occur on the same simulation delta cycle. Because such systems are not centrally controlled and VHDL signal resolution functions cannot resolve the time order of signal transactions, tokens may collide with undesirable results. Various means can be used to avoid the unintentional annihilation of a meaningful token. For instance, we have found that implementing a simple delta-delay first-in first-out (FIFO) queue process on each port is useful. Such a FIFO process introduces zero time delay but places tokens onto the port on sequential delta cycles if more than one token is assigned to the same port on the same delta cycle by current processes within the SE entity.

We have found the VHDL-'93 shared-variable feature to be useful in these abstract models for registering state information, which can be accessed by concurrent processes within an entity. Such variables allow a more elegant and robust abstraction of a component that must allocate resources among multiple, competing processes. A variable representing an available resource may be tested by a requesting process and set atomically to allocate the resource before another process tests it because variables take on newly assigned values immediately. The alternative method of using local signals can cause a race condition, because signals do not take on their newly assigned values until the delta cycle changes.

4 Performance Metrics

Considering the purpose of performance modeling, there are several measurements that are obtained from the simulation of the models. These include processing and communications latencies; throughput; event timelines; utilizations of memories, processors, and links; and other statistics. Such statistics can be captured by placing monitoring statements in the appropriate places within the VHDL models. For instance, the following variables record the PE utilization and memory allocation respectively:

```
-- Declarations for performance stats
VARIABLE pe_use           : TIME := 0 ns;
VARIABLE memory_allocated : INTEGER := 0;
```

The first variable can then be used to record processor utilization by placing a statement such as the following near the compute delay of the PE model:

```
-- Integrate the processing time
pe_use := pe_use + compute_delay;
```

The second variable can be used to record the memory allocation by incrementing and decrementing it whenever simulated memory is allocated or freed, respectively.

Timeline information can be captured by placing text-I/O statements in the models that write the time and name of relevant events to a history file. The event history can then be visualized by post-processing the event file to produce a two-dimensional plot. We have found that various time-line plots are very useful in understanding the composite behavior of a system. For instance, figure 4 is an example processing timeline plot that shows when specific PEs are busy processing specific tasks. Similar timeline graphs can be made that show when PEs are idle; when PEs are sending or receiving data; or when communication links are in use.

Plots of quantities, such as memory allocation can be produced by post-processing the event history file of a performance simulation, as shown in figure 5.

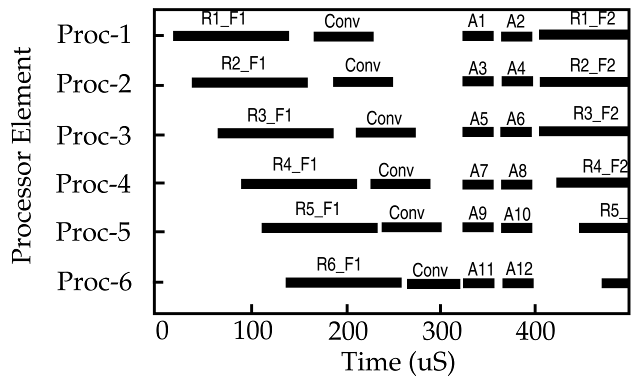

Figure 4 - Example processing time-line.

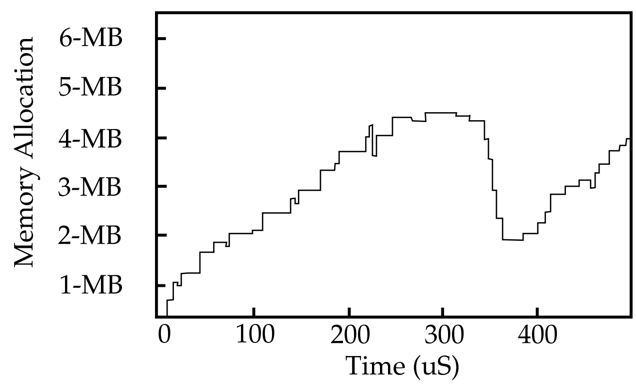

Figure 5 - Example memory allocation time history.

5 Virtual Prototype

A virtual prototype model of a DSP system can be quickly realized from the type of performance model described above. The principle means to extend the performance model is to augment the interpretation of the *compute* instruction in the PE-model with the associated mathematical functions, and to allocate and maintain a local memory array for storing the actual data values in the PE and ME models. The token definition must be extended to include a data field for carrying the actual data values over the modeled network. The SE model need not be modified, but it needs to be re-compiled due to the modification of the common token definition.

When the above modifications are made, the resulting virtual prototype model is data, function, and timing accurate. It can not only produce time-line and performance measurements, but it can process input data and produce correct output data at the appropriate times. We say it is a virtual prototype because it appears to react to applied stimuli in value and simulated-time as we expect the target system to react. The virtual prototype is a useful development tool because it allows non-invasive study of the DSP operation. We can stop the virtual prototype at any time to examine what values are held in each part of the system. We can also step through the execution slowly to compare observations with the target system during development.

In our experience, a VHDL virtual prototype model tends to execute less quickly than a VHDL performance model of similar abstraction level. The largest slow-down is due to the copying of the data records on token assignments to signals, which requires more time for the augmented token of the virtual prototype. In accordance with the VHDL Language Reference Manual, pointers to data cannot be passed between entities, but data values are passed by value.

6 Observations and Conclusion

Simulations must execute and return results from simulation runs in roughly an hour or less to permit designers to rapidly explore, optimize, and verify DSP 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 rather than months.

The techniques described above were used to simulate a 24-PE system executing 5 seconds of a synthetic aperture radar application[6]. The system was modeled with VHDL at the network-performance level. The simulation consumed 28 Sparc-10 CPU minutes. When considering the number of PEs modeled and their instruction rate, the effective execution rate of the simulation was approximately 2,857,000 instructions-per-second.

The virtual-prototype model of the same system and simulated duration consumed approximately 14 CPU-hours and exhibited an effective execution rate of 23,810 instructions per second.

At an even lower abstraction level, the execution rate of a typical ISA-level VHDL model of the same i860-based PE exhibits an order of 5.5 to 7.5 instructions-per-second on a Sparc-10 CPU [7]. Such ISA models are useful for understanding the behavior of software segments that operate local to a given PE.

The three VHDL model types discussed above are each useful for specific tasks in the design process. The range of effective execution rates underscores the need to carefully select the appropriate abstraction level for the design task at hand. The ability to successfully model complex systems at many levels of abstraction highlights the elegant design and foresight with which the VHDL language was devised.

Acknowledgments

We thank Bill Kline for his many comments and assistance in exploring VHDL modeling methods.

References

[1] Rose, F., et. al., "Evaluating Distributed Multiprocessor Design", Proceedings 2nd Annual RASSP Conference, Arlington, VA, July, 1995.

[2] 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.

[3] Yang, J., Bic, L., "A Mapping Strategy for MIMD Computers", Proceedings of the 1991 International Conference on Parallel Processing, April, 1991.

[4] 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.

[5] Lin, X., McKinley, P., "Performance Evaluation of Multicast Wormhole Routing in 2D-Mesh Multicomputers", Proceedings of the 1991 International Conference on Parallel Processing, April, 1991.

[6] Zuerndorfer, B., Shaw, G., "SAR Processing for RASSP Application", Proceedings of the RASSP Conference, August, 1994.

[7] Egolf, T., et. al., "Experiences with VHDL Models of COTS RISC Processors in Virtual Prototyping for Complex System Synthesis", Proceedings of the VHDL International User's Forum, April, 1995.