Using VHDL for High-Level, Mixed-Mode System Simulation

The computational and I/O requirements of complex computer applications such as embedded control, signal processing, and automation have necessitated the development of systems with architectures tailored specifically for particular applications. These application-specific systems are usually complex, distributed, concurrent systems involving a mix of dedicated and general-purpose hardware and software. Spanning multiple packaging levels—chips, multichip modules, printed circuit boards, backplanes, and even networks—they usually interface with a variety of sensors, actuators, and other electromechanical components. Integrating these heterogeneous hardware, software, and electromechanical components to form a complete system presents a challenging design problem.

CAD research has concentrated on the design of individual application-specific ICs (ASICs). Progress in CAD tools has reduced ASIC design time dramatically. In some application domains, it is even possible to completely synthesize an ASIC from a high-level behavioral description in a matter of hours.\(^ 1\)\(^ 2\) In short, ASIC design is no longer a bottleneck in the system design process. But ASICs alone do not constitute a system—they are just part of the system. Unfortunately, CAD development so far has largely avoided taking a holistic approach to the design process.

Motivated by the lack of system-level design aids, we have developed SIERA (system integration of embedded real-time applications), a CAD framework for the rapid prototyping of application-specific systems. Figure 1 on the next page shows a global overview. SIERA implements a high-level system specification as a set of custom printed circuit boards using a mix of software-programmable and non-programmable hardware and ASICs.

A key step in the rapid-prototyping process is the simulation of the high-level user specification of the system, as shown in the top left portion of Figure 1. Traditionally, designers have performed simulation at this level either with specialized high-level simulation languages such as Csim\(^ 3\) and Aspol\(^ 4\) or in an ad hoc fashion by writing custom simulators in C or some other programming language. VHDL enables simulation across many different levels of abstrac-
tion, and its process-oriented viewpoint makes the expression of concurrence elegant. These advantages, and its increasingly wide acceptance as a standard, motivated our using VHDL in SIERA to enable a designer to simulate the high-level system specification and the physical environment that interacts with the system. This is a somewhat non-traditional application of VHDL, which so far has been used primarily for relatively low-level hardware modeling.

Given the unconventional nature of our application, we expected, and indeed encountered, situations where we had to use VHDL in a convoluted fashion and suffer performance penalties. Nevertheless, VHDL's ability to allow multilevel simulation—simulating different parts of the system at different levels of detail—is a strong enough motivation to do the high-level simulation in VHDL as well.

Example system

We are using our framework to develop several complicated systems, including a multisensory robot controller. This application illustrates well the issues involved in high-level system simulation. Figure 2 shows the architecture of the robot controller, a real-time system that controls a six-degree-of-freedom robot arm by means of position, force, and vision sensors. The system is based on a set of custom boards for vision sensing, motion control, and actuator interfacing. The interconnected boards are also linked to a single-board computer and a workstation.

The robot system exhibits an interesting mix of general-purpose and custom hardware and software, digital and analog hardware, electromechanical components, and interaction with the environment. These attributes are present in varying degrees in most application-specific systems. We can best describe the robot arm and the analog subsystems with sets of differential equations. We can best model the software subsystems as a set of processes communicating via message-passing queues or shared data structures. Similarly, we can model the dedicated digital hardware subsystems as a set of processes, but with different interprocess communication (IPC) primitives. It is this diversity of behavior that makes modeling applicationspecific systems extremely difficult.

In our investigations, we identified two key problems in using VHDL for high-level behavioral simulation of a system like the robot controller: the VHDL signal's unsuitability as the primitive for system-level interprocess communication and the difficulty of modeling continuous-time dynamic subsystems such as the robot motors.

High-level system modeling

A model specifies and expresses a system's behavior by abstracting its salient properties. Computer designers and engineers use models for design, simulation, analysis, and verification. Even in the domain of digital computing systems alone, researchers in fields such as computer-aided IC design, computer-aided software engineering, and digital signal processing (DSP) have developed a plethora of diverse models. Their diversity stems primarily from one or more of the following factors:

- **application domain:** general-purpose software systems, ASICs, DSP systems
- **modeling goal:** rapid prototyping, verification and validation, analysis
- **underlying computation model:** sequential imperative, functional multithreading, communicating-process formalisms

No single model is suited for all applications. For example, DSP applications usually are best modeled as signal flow graphs. In contrast, control-dominated applications, such as communications protocol processors, are best described as communicating finite-state machines. Most systems in the class of interest to us—dedicated, real-time systems that continually interact with the environment—are naturally expressed at the
top level as a set of processes that operate concurrently and communicate with each other and the environment. For example, the architecture of many robot systems consists of communicating sequential processes. Furthermore, the underlying hardware in such systems is inherently concurrent. It consists of multiple, general-purpose, software-programmable processors combined with dedicated hardware devices that serve as computation accelerators or I/O peripherals. The hardware devices operate in parallel with the software-programmable processors, communicating and synchronizing with them. We can represent these hardware devices at the software level by treating them as separate processes running on their own dedicated processors.

Many researchers have realized the utility of describing a system as a set of cooperating processes. They have proposed many system models that can be loosely classified as graphs in which nodes represent concurrent computing agents, or processes, and edges represent paths along which the processes communicate. These models differ most in their IPC mechanisms, the allowable behavior of individual processes, and whether the set of processes is static or dynamic.

VHDL's underlying model belongs to the same category. VHDL represents a system as a static set of sequential processes that communicate and synchronize via signals with well-defined semantics. Unfortunately, as other writers have pointed out, the VHDL signal is not a suitable IPC primitive for high-level system specification and modeling. Message-passing channels, events, and shared-data structures are the more commonly used IPC and synchronization primitives.

The VHDL signal is not suitable for high-level modeling for two reasons. First, although a last-driven resolution is very useful for high-level modeling, VHDL allows only static resolution in case of multiple drivers. Second, modeling asynchronous communication with a VHDL signal is difficult. For example, although we can represent memory in VHDL (by using guarded register signals), VHDL signals do not easily provide the queue buffer memory needed.
for high-level modeling of asynchronous communication.

Several recently published techniques indicate the complexity of high-level modeling in VHDL. For example, Aylor et al. describe an elaborate VHDL technique based on extended Petri nets. In this technique, a handshake protocol built on top of VHDL signals implements the token-passing mechanism.

**Alternative IPC models.** We can classify the various concurrent-process models along the following dimensions:

- **Process set**
  - Is it static or dynamic?
  - Is it bounded or unbounded?

- **Behavior of processes**
  - Do processes have state?
  - Can processes exhibit indeterminate behavior? (For example, can a process wait for one of two inputs, whichever arrives first?)

- **IPC mechanism**
  - Is it based on shared memory or message passing?
  - Is the message passing synchronous (unbuffered) or asynchronous (buffered)?
  - Is the buffer depth bounded or unbounded?
  - Is the information flow unidirectional or bidirectional?
  - If bidirectional, is it simultaneous or delayed?
  - Is the communication control symmetric or asymmetric?
  - Is the message channel FIFO ordered?
  - Can the channel have multiple readers or writers (that is, can multiple processes read from or write to the channel)?

Using these dimensions, we describe some of the popular system specification models:

**CSP.** The CSP (communicating sequential processes) model represents the system as a static set of possibly nondeterministic, sequential processes that communicate with each other via unidirectional, unbuffered (synchronous) communication channels with one reader and one writer.

**Functional multiprogramming.** Functional multiprogramming models the system as a static network of deterministic, sequential processes that communicate via unidirectional FIFOs, with infinite buffering and a single writer and multiple readers. The network can have unbounded recursive parallelism. The processes cannot wait for input on more than one channel simultaneously. Mathematically, such a sequential process is equivalent to a continuous function from the sequences on input channels to the sequences on output channels. This model is a progenitor of various dataflow models. It can model only deterministic computation in which the result does not depend on the order of computation.

**Capsim.** Capsim is a block diagram simulator for DSP systems. Its specification methodology is a functional multiprogramming subset in which the process set is bounded and the channels are restricted to single readers.

**Static dataflow.** The static dataflow model too is a subset of functional multiprogramming. The set of processes is bounded, the channels are restricted to single readers, and they are nonbuffered. Asynchronous hardware techniques, recently popular, also follow the static dataflow model.

**Hardwaredc.** Proposed for behavioral specification of ASICs for synthesis, Hardwaredc allows a static set of processes that communicate through unbuffered channels as in CSP or through a shared medium.

**Caim.** Caim models a system as a dynamic set of processes communicating through events and message queues that can have multiple readers and writers. The events are like buffered channels of depth one in which the writer overwrites the buffer if it is full. The message queues are infinitely buffered FIFO channels.

**Silage.** A language for DSP system specification, Silage employs the signal flow graphs used in signal processing. It is distinctive principally for its syntax; it is otherwise isomorphic to the static dataflow model.

**Our model.** The following requirements motivated our choice of a high-level model for application-specific systems. First, it must be able to represent the kind of application-specific system we are most interested in—an embedded system that continually interacts with its environment in real time. Second, it should be suitable for rapid prototyping of the system as a mix of dedicated and general-purpose hardware and software. Third, the model should be easy to simulate. Fourth, its behavior should be easy to analyze, so that CAD tools can manipulate and transform the specification for synthesis. Finally, the model should encourage modular composition of the system, reusability of system-level modules, unified treatment of hardware and software modules, and uniform communication between modules.

The model we adopted for our rapid-prototyping framework is the process network model. It views an application-specific system as a static, hierarchical network of sequential processes that execute concurrently and interact by means of a well-defined communication mechanism based on FIFO channels. Each process has several input and output ports, and a channel connects an output port to an input port. A process can send data samples at an output port, receive samples at an input port, or wait for one of a...
set of ports to become ready for communication. We can represent the process network as a block diagram, a very natural paradigm for thinking about the class of systems of interest to us.

The model characterizes channels by a buffer depth and a data type. A buffer depth of zero indicates synchronous or unbuffered communication. A buffer depth greater than zero indicates asynchronous communication. Infinite buffer depth is allowed for simulation. The model characterizes ports by a data type and a port protocol. Port protocols specify the behavior of a process when a channel is full or empty. For an output port, the protocols are block on full, overwrite on full, and ignore on full. For an input port, they are block on empty, previous on empty, and ignore on empty. The port protocols are fixed and cannot change dynamically. The model includes the overwrite on full and previous on empty protocols to handle many low-level hardware interfaces.

In terms of the classification just presented, the process network model has a bounded, static set of processes that can have state and that exhibit indeterminate behavior. The IPC mechanism is based on unidirectional, FIFO-ordered, synchronous or asynchronous message-passing channels with one reader and one writer.

The process network model views the sensors, actuators, and mechanical subsystems, which allow the system to interact with the physical world, as preexisting processes executing on their own dedicated electrical or mechanical hardware in parallel with the rest of the system. The model thus provides an abstract representation of the entire system and its environment.

**VHDL for high-level simulation**

A model for specification of application-specific systems is not particularly useful without a simulator to execute it. One strategy would be to use a dedicated simulator or a simulation language specialized for high-level simulation. The disadvantage of that strategy is that it rules out selective modeling of parts of the system at lower levels of detail. Instead, we decided to simulate our model of high-level system specification on top of a VHDL substrate, thereby integrating a multilevel simulation environment into our rapid-prototyping framework.

To carry out our plan, we wrote two sets of VHDL packages that provide primitives suitable for high-level system simulation. One set supports modeling of the IPC mechanism used by our process network model. It also supports shared-memory-based interprocess communication and stochastic modeling, both often useful in high-level modeling. The second set of packages supports modeling of continuous-time subsystems such as sensors and actuators, for which VHDL does not provide direct support.

In the following discussion, we often refer to template packages. The term refers to a preprocessing strategy we have used to bypass the limitations imposed by VHDL's lack of generic packages (packages parameterized by a data type) and its lack of the function pointer type. The user generates instances of a template package, using a simple preprocessor that customizes the package for a particular user-defined data type or function by means of text substitution. So far, we have found template packages to be the cleanest way to overcome the VHDL limitations.

**VHDL modelsimulating package.**

A VHDL template package called MSGPACK_type provides primitives to support system modeling according to the process network model. It allows VHDL processes to communicate using special signals that implement the FIFO, channel-based communication mechanism. The template package is parameterized by the data type type carried by the channel, and the user must generate an instance of the package for each channel data type.

MSGPACK_type implements channels as separate VHDL entities that connect with ports in the source and the destination processes, using signals of a special data type called MSG_type. One such channel entity must be used for every channel in the process network. When the user inputs the process network, using a special parameterized structure description language, our framework automatically supplies the channel entity. A single VHDL generic for specifying the buffer depth parameterizes the channel entity. The channel entity has a third port, where it outputs data about the channel state, to provide statistics about the data traffic through the channel.

The VHDL processes that model the computation nodes in the process network model access a channel through msgSend( ) and msgRec( ) operations on the corresponding VHDL ports of type MSG_type. Processes can perform these operations in one of several modes corresponding to the port protocols mentioned earlier. For writing to an output port, the permissible modes are block on full (BMODE), overwrite on full (OMODE), ignore on full (IMODE), and error on full (EMODE). For reading from an input port, the permissible modes are block on empty (BMODE), previous on empty (PMODE), ignore on empty (IMODE), and error on empty (EMODE). The error on full/empty mode accommodates simulation only.

Two other important procedures are waitAny( ) and waitAll( ). They take a list of input or output ports and return only when any or all of them are ready for communication. Several other utility routines enable monitoring of the channel state. These routines are meant strictly for simulation.

The key issue in writing the MSGPACK_type package was the implementation of channels, as defined by the process network model, using VHDL signals. As indicated earlier, a separate VHDL entity models the channel buffering. VHDL signals of type MSG_type con-
Figure 3. VHDL implementation of channels in the process network model.

nect the buffer to user processes. This data type is really a composite type, and we can view it as a bundle of primitive VHDL signals. Besides the signal to carry the data sample, the bundle includes signals to implement a four-phase handshake between the buffer process and the sender or receiver process. In addition, MSG_type includes signals to transmit channel information to the sender or the receiver. Figure 3 shows a simplified picture of the primitive signals that constitute an MSG_type signal and lists the functions available to the sender and the receiver processes.

The ports of type MSG_type must be of mode inout, because some of the control signals that implement the handshake move in a direction opposite that of the data. Every MSG_type signal, therefore, has two ports of mode inout connected to it—one from the buffer side, the other from the side of the sender or receiver process. Implementing the handshake requires a complex resolution function. The complexity arises from the need to disambiguate in the resolution function which driver is connected to the buffer and which to the sender or receiver. This connection may take place through a series of post-signal associations, complicating things even further. We use an algorithm that maintains information about how far a signal is from the ports in the buffer process and the sender or receiver processes. The resolution function uses this information to identify the drivers.

We developed the MSGPACK_type package to simulate the process network model in our rapid-prototyping framework. But with an appropriate combination of channel type, write mode, and read mode, designers can also use it to simulate many similar system models, some of which were discussed earlier. Such simulations must be carried out under the constraints of a static set of processes and channels with a single reader and writer.

Shared memory IPC. In many modeling situations, using shared memory as a form of interprocess communication is a quick and dirty, but efficient, technique. Although the MSGPACK_type package can express shared memory, it is not very efficient. Therefore, we wrote a special VHDL package called SHM PACK, which allows modeling of shared data objects. The shared memory behaves like a true multiprotocol memory with one object of the specified data type. A VHDL process uses procedures shmRead() and shmWrite() to perform an atomic read and write on the shared data object.

Shared memory inevitably leads to resource contention, and a low-level solution to this problem is binary mutual-exclusion semaphores. Another VHDL package, SEMPACK, implements two flavors of such a semaphore—one with a priority scheme for waking up waiting processes and another with a first-come-first-served scheme. The SHMPACK and SEMPACK packages together let the user express most shared-memory situations. The implementation details of these packages are described elsewhere. Figure 4 shows simplified headers for the two packages.

Modeling continuous-time subsystems. Sensors, actuators, electromechanical components, and analog electronic subsystems are integral to many applications-specific systems. Like those in the robot system, they are all continuous-time, dynamic systems. To perform a meaningful high-level simulation of the robot controller, we must simulate its operation along with the servoamplifiers, the robot arm, and the system’s physical environment. Unfortunately, VHDL does not provide any built-in support for such modeling.

Recent work has investigated SPICE-like, low-level simulation of analog circuits in VHDL. Such simulation, however, is not very useful for high-level system modeling. We require high-level behavioral modeling of commonly encountered continuous-time subsystems. A set of coupled, nonlinear, time-varying, ordinary differential equations (ODEs) can easily express such behavior. In many cases, such as the robot arm mechanics and the electric motors driving them, a set of ODEs is the most natural representation.

There are two possible approaches to modeling with ODEs. One is to link a VHDL simulator to a simulator such as Simmon, which is capable of simulating dynamic systems expressed as ODEs. Multisimulator integration environments such as Ptolemy, combined with foreign-simulator interfaces provided by many VHDL simulators, may make this an attractive solution in the future. We have taken a more tradition-
Figure 4. VHDL packages for shared memory and binary semaphores.
The following state-space formalism represents the dynamics of each continuous-time subsystem:

\[
\begin{align*}
\dot{s} &= f(s, u, t) \\
\dot{u} &= g(s, u, t)
\end{align*}
\]  

(1)  

(2)

where \( s \) is the state vector, \( u \) is the input vector, and \( \dot{u} \) is the output vector.

To model each subsystem, a VHDL process solves these equations numerically by repeating two distinct steps. Using Equation 1 and standard numerical techniques, the VHDL process calculates the state vector at time \( \Delta T \) into the future. It may take a series of small, adaptive time steps to accomplish this. \( \Delta T \) itself depends on the dynamics of the model as well as the dynamics of the inputs. After advancing the time, the process recalculate the output vectors, using the new state vector and the current input vector. This step is not as simple as it appears. The problem is that the different continuous-time blocks update their outputs asynchronously. These outputs may form inputs to other blocks in the coupled system.

We must make sure that all these signals stabilize at a given time step before the time is advanced. To accomplish that, we use the following algorithm: Each subsystem reevaluates its outputs every time any of its inputs changes at a given time step. If there are no algebraic loops in the interconnected system, the dataflow graph (formed by Equation 2) of all the subsystems is acyclic, and this process is guaranteed to terminate. If there are algebraic loops, the process is equivalent to a distributed relaxation algorithm for solving a nonlinear fixed-point problem, and convergence is no longer guaranteed. The presence of such algebraic loops is usually an indication of poor modeling. The following simplified pseudocode demonstrates the preceding algorithm. The procedure \( f(\cdot) \) uses the derivative function \( f' \) to calculate the state at the next time step.

\[
\begin{align*}
t &= \text{NOW}; \\
\text{tnext} &= t + \text{deltaT}; \\
\text{L1: while TRUE loop} \\
&\quad \text{solve for output vector } u \text{ at the current time instant } t \\
&\quad \text{using a distributed iterative algorithm} \\
&\quad \text{L2: while TRUE loop} \\
&\quad \quad v < = g(s, u, t); \\
&\quad \quad \text{usave} = u; \\
&\quad \quad \text{wait on } u \text{ for deltaT}; \\
&\quad \quad \text{exit L2 when NOW > t; exit L1 when t = tnext; } \\
&\quad \text{end loop L2; calculate state at tnext} \\
&\quad \quad \text{odesolve}(s, s, t, tnext, \ldots); \\
&\quad \text{if NOW < tnext then} \\
&\quad \quad \text{wait for tnext - NOW; } \\
&\quad \quad \text{end if; } \\
&\quad \quad t = \text{tnext}; \\
&\quad \quad \text{tnext = NOW; } \\
&\quad \text{end loop L1;}
\end{align*}
\]

The only remaining problem is to calculate the state vector at time \( \Delta T \) into the future. Many numerical techniques are available in the literature for this. We use a fifth-order Runge-Kutta method with monitoring of local truncation error to adapt the step size.

Implementing VHDL packages. We have implemented the strategy outlined in the previous section in VHDL. VHDL's lack of generic or template packages and function pointers makes an elegant implementation difficult. As shown by Equation 1, the ODE solver routine inherently depends on the derivative function \( f'(\cdot) \), making it impossible to write a single VHDL package that works everywhere. The user must customize any continuous-time modeling package in accordance with \( f' \). To overcome this problem, we wrote a template package called \( \text{RK}_{< \text{derivs}>} \), from which the user can generate a particular instance of the package for a given \( f' \), using a simple preprocessor program.

The template package implements in VHDL the Runge-Kutta subroutines described by Press et al. It makes two functions available to the user:

\[
\begin{align*}
\dot{s} &= A \times s + B \times u \\
\dot{u} &= C \times s + D \times u
\end{align*}
\]  

(3)  

(4)

The package \( \text{ODE_linderiv} \) provides functions \( \text{odesolve_rk4 clums_linderiv} \) and \( \text{odesolve_rk4smart_linderiv} \). Using these functions, a VHDL entity called \( \text{LTI} \) implements a generic, continuous-time, linear time-invariant block. The user passes the constant coefficient matrices \( A, B, C, \) and \( D \) as generics to \( \text{LTI} \). Figure 5 shows the use of \( \text{LTI} \) in modeling a DC motor.

To illustrate how the various packages we have discussed are used in a complete high-level system simulation, Figure 6 presents a simplified high-level block diagram of the robot control system. Even such a simple system consists of blocks that require dramatically different modeling techniques. We model the DC motor as a linear, time-invariant, continuous-time system, whereas we describe the robot arm (single joint) by means of a set of nonlinear, time-varying differential equations. On the other extreme, we describe the planner, the trajectory generator, the front end, and the
controller in terms of processes and IPC channels. The optical encoder and the digital-to-analog converter and amplifier are hybrids because they interface between the discrete-event and the continuous-time parts of the system.

We modeled portions of the robot system, using the described packages, and simulated them on the MCC (Microelectronics and Computer Technology Corporation) VHDL simulator. However, a simulation of the entire system remains impractical because the model’s complexity requires exorbitant simulation time. With the techniques we’ve described, however, one can easily accomplish high-level simulations of similar but less complicated systems. In addition, VHDL’s native capability for low-level digital hardware simulation makes it easy to model the low-level system hardware.

VHDL presents many syntactic as well as semantic obstacles to doing high-level system simulation elegantly. Nevertheless, a VHDL substrate offers the tremendous advantage of mixing high-level simulation with low-level simulation—not feasible with dedicated high-level simulators. This capability supports a powerful, multilevel, mixed-mode simulation environment.

**Acknowledgments**

The US Department of Defense Advanced Research Projects Agency funded the work we’ve described. We thank Brian Richards.
References


Mani B. Srivastava recently received his PhD from the University of California, Berkeley, and now works in the Network Computing Research Group at AT&T Bell Laboratories, Murray Hill. His research interests include CAD tools for hardware-software prototyping for board-level systems, design of custom VLSI and board-level hardware, and architecture for real-time systems in digital signal processing and robot control. Srivastava received the BTech degree in electrical engineering from the Indian Institute of Technology at Kanpur, India, and the MS degree in electrical engineering and computer sciences from the University of California, Berkeley. He is a member of the IEEE Computer Society.

Robert W. Brodersen is a professor in the Electrical Engineering and Computer Science Department of the University of California, Berkeley. In addition to teaching, he researches new applications of integrated circuits focusing on signal processing and related CAD tools. He is also an industrial consultant on signal processing applications, CAD tools, and analog and digital IC design. Previously, he worked for the Central Research Laboratory at Texas Instruments, on the operation and applications of charge-coupled devices. Brodersen received BS degrees in electrical engineering and in mathematics from California State Polytechnic University, Pomona. He received the Engineer's degree, the MS degree, and the PhD degree from the Massachusetts Institute of Technology. Brodersen is a fellow of the IEEE and a member of the National Academy of Engineering.

Send correspondence about this article to Mani B. Srivastava, AT&T Bell Laboratories, 600 Mountain Ave., PO Box 636, Murray Hill, NJ 07974-0636; mbs@research.att.com.