

VHDL: Is the Phoenix Burning?

Cary Ussery

Alta Group of Cadence Design Systems

270 Billerica Road

Chelmsford, MA 01824

cary@cadence.com

Abstract

This paper explores the challenges for system-level design modeling and verification and examines the applicability of VHDL to them. Fundamental issues are raised in which VHDL currently falls short of addressing these challenges in an effective way. Among these are interprocess communication semantics, evolution of design blocks through the design process, distinction between system function from system architecture, and the separation of behavior and communication. We then explore language requirements for addressing these challenges. Implications on VHDL and, in particular, on a potential evolution of the language are raised. Finally, we conclude on recommended future paths for VHDL.

Introduction

Until recently, system architects and system implementers operated in different domains. System functionality and architecture was defined and verified using lightweight representations and languages and then the architecture was communicated, usually in a written document, to the design and implementation community. With deep-submicron technology, complete systems functionality can be implemented on a single IC. This, coupled with time-to-market pressures on electronic systems design teams, requires a closer relationships between system architects and system implementers.

A new design flow is required to transform a design from its functional specification into working silicon. In particular, electronic design is evolving toward a methodology to design ICs by integrating multiple, usually pre-built, on-chip blocks of “intellectual property” (IP). We identify four major levels in the flow that are particularly appropriate for IP--based flows but apply as well for the future of HDL-based design: System, Mapping, Architecture, Implementation.

The characteristics of these four stages are:

FUNCTIONAL

This stage is for checking out the validity of the algorithms and other functionality chosen for the product. Models are

typically developed in either block-level graph environments or in C, C++ and similar languages.

ARCHITECTURAL

In this stage, the algorithm is mapped onto a set architectural “resources” whose elements may be concurrent and overlapping duty cycles and which may be chosen for implementation either as hardware or software for one or more instruction-set processors. Modeling at this level has traditionally been difficult. The most successful approaches have utilized discrete-event, queue-oriented systems to analyze the performance characteristics of the system.

ASSEMBLY

In this stage, the mapping is completed by a representation that is amenable to direct translation to implementation. HDLs at the behavioral and RTL levels will be used to generate cycle-accurate models which can, in many cases, be synthesized into logic. In a block-based flow we expect heavy use of predefined large blocks and block descriptions that are transformed by special-purpose generators. We observe that in both flow classes a particularly emerging block-based flows, the HDL that is used comes from libraries or generators from higher level Software at this level is typically in C or C++ although we expect to find increasing use of Java for embedded systems.

MANUFACTURING

This stage produces the “sign-off-ready” representation of the product with which the functional tests that remain after verification at the preceding levels may be carried out. In particular, timing-related verification occurs at this level. Shi et al.(1966) make the point that verification models at these levels run progressively slower as design passes from system to implementation. In his case

system: mapping: architecture: sign-off
1:100:18000:27000

although the correspondence between his categories and ours is approximate.

We believe that HDL-based flows will continue to enjoy investment and growth for a few years to come but that they will not be able to rise to the challenge of rapid and reliable delivery of the complex chips that the consumer market is demanding. Their use, and the importance of HDL-based design, will therefore diminish in favor of IP-based design approaches, although there will continue to be roles for these flows for small chips (< 500K gates) and for some of the authoring process required by IP-based design.

The rest of this paper will focus exclusively on the system level and specifically look at why VHDL in its current form is not appropriate for modeling at this level.

System level design and verification

System level descriptions are used to capture the essential functionality, major algorithms and environment in which the system operates. A system description can be viewed as a combination of :

- functionality or behavior of the system in response to its environment
- system boundary for interacting with its environment
- static and/or dynamic models of the environment in which it operates
- constraints on the implementation of the system

System level designers are faced with a number of key issues. In particular, they must model three key aspects of the overall system: the algorithm or functionality, the protocol(s) through which the system interacts with the outside world, and the environment in which the system must operate.

Algorithm Design

During algorithm design, design teams are usually concerned with the impact of given algorithmic decisions on the capability or quality of the product under design. For instance, designs teams might try to assess the impact of a new compression technique on the audible sound produced by the system or evaluate error rates with changes in a signal-to-noise characteristic of a system. Such analyses require applying real-world (or representative) data to the system and examining the dynamic reception, processing, and response of the system to that data.

Algorithms are verified by running sequences of data through them. Typically, design teams are interested in the impact of algorithmic decisions on system quality. For

instance, design teams might want to evaluate the effect of different signal-to-noise ratio characteristics on the quality of a processed audio signal. To do this, the design team generates real-world data for the input audio signal, applies that data to the algorithm using simulation, and saves the resulting audio data. This data can then be analyzed heuristically by “playing” the generated audio data and quantitatively by view the signal waveforms for the generated audio. This involves applying streams of audio data (waveforms) and generating results in near real-time.

Protocol Conformance

The word protocol here refers to the communication protocols through which the system interacts with its environments. In wireless communications, this includes GSM or IS-95 using lower level standards such as CDMA or TDMA. In telecommunications, this includes standards like ATM. A system is expected to accept and transmit data using these standard protocols within predefined performance windows. Today’s systems are often faced with supporting multiple standards or products must adapt to geographically distinct protocol standards. Products often must pass regulatory type approval (e.g., FCC or ETSI type approval) before being allowed into the market. If a product does not conform to a set of well-defined protocols it will not pass type approval. This can mean significant lost opportunity and revenue for the product.

Environment Modeling

One of the most difficult tasks in system-level modeling is to model the environment in which a system must operate. Environments for modern day electronic systems are complex and include multiple types of data (audio, video, etc.) being distributed in multiple formats (MPEG, JPEG, etc.) over multiple communication protocols (ATM, CDMA, etc.). In addition, systems are faced with highly volatile contexts in which they must operate. For instance, mobile systems must be able to maintain and optimize signal transmissions while the system is moving between different cells in the overall wireless network. To effectively evaluate the capabilities and correct functionality of a mobile handset design, the design team needs to provide real-world audio data in the right formats to the system and provide real-world interaction with the transceiver base station including the effects of moving through the cell network.

It is also useful, to analyze the results of the system processing both in detailed representations and in more user-focused representations. For instance, design teams would like to analyze audio quality both through hearing the results (user-focused) and utilizing audio waveform

analysis tools. This allows the team to cross-correlate the actual impact on audio quality for given algorithms and algorithmic decisions.

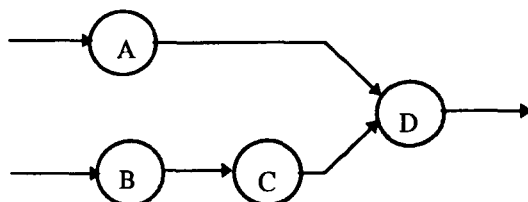
Design Representation

System functionality (including algorithms, protocols and environments) are generally described as a collection of distinct communicating processes or process networks. These processes are viewed as executing concurrently with communication between them managed through some model of communication. System functionality is often separated into the algorithmic processing functions and the control functions of the system. These two aspects of the system have different needs in terms of their communication model.

Data Flow Descriptions

Algorithmic processing is usually defined as a set of operations operating on a stream of input data and producing a stream of output data. The communication model most commonly used here is data flow semantics; i.e., a process (data flow actor) fires when new data (tokens) is available at each of its inputs. There is no notion of time in a data flow simulation; the goal is to transform a stream of input data into a stream of output data.

A key aspect of efficient data flow simulation is the scheduling of nodes in the process network. Processes execute data as their input tokens become available. The order in which processes must execute to ensure that their inputs are available is often derivable from the connectivity of the process network. For instance, consider the following diagram. It is clear that both process A and



process C need to be executed before D will be ready to execute. In addition, process B must execute before process C. A schedule of A->B->C->D will ensure that inputs are available when processes are executed.

Control Flow Descriptions

Control aspects at the system level are used to model the way the system responds to some external stimulus or event. Here again we use process networks. However, the behavior of these networks is quite different than that of

data flow graphs. Typically, the action a process takes when it is fired is often dependent on the current state of the process. In addition, processes in control flow networks have more varied firing mechanisms; they may fire when an event (token) is detected on any one of their inputs or on some conditional combination of tokens on their inputs (including waiting for tokens on all their inputs as in data flow networks). Often, the actual process can be viewed as a finite state machine which reacts to input events by emitting new control actions for the system.

Communication Modeling

If the system description contains multiple process networks, the communication between them must be modeled. The communication path between two process networks is often represented by a queue of length 1. This essentially provides a buffering of data which is moved from one process network to the other. Queues of different lengths can be tried to understand the communication requirements within the system. For instance, a data flow network might operate on four data inputs but the control network might be designed to produce these inputs one at a time. In this case, a queue of length 4 is required to enable the proper communication between them.

Many times the communication between two networks is dependent on the state of the system or of the communication channel itself. Consider, for instance, a communication channel which might be interrupted during the execution of a data transfer. In this case, the communication channel itself can be modeled as a control process network.

VHDL Issues

At the surface, none of these requirements prohibit the use of VHDL for system level modeling. However, by looking just below the surface a number of major issues emerge.

Interprocess Communication

One of the key aspects of a system model is viewing the system as a collection of concurrent processes. The overall system description is used to explore and define the way in which these concurrent processes interact with each other.

In VHDL, an elaborated design creates a network of concurrent processes. Processes interact with each other through signal networks. The designer has only limited control over the characteristics of these signal networks. In particular, the delivery of a signal event to a process sensitive to that signal is limited to transport or inertial

semantics. The designer can only insert type conversions or bus resolution functions into the signal network between processes.

System level descriptions are often concerned with modeling the communication semantics between processes. For instance, queuing models are often used to analyze the eventual buffering requirements between system functions which operate at different rates.

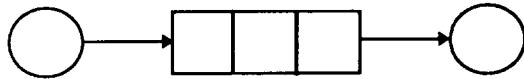


Figure 1 - FIFO Queue of Length 3

There is no effective way of modeling this queue using the communication semantics of VHDL. The only approach is to model the queue as a component itself. However, even then, the designer would need to add significant processing and handshaking signals/ports to control simple tasks such as blocking the left hand process when the queue is full or controlling reading off the queue from the right hand process. (The skeptical reader is encourage to try writing a robust queue component at home).

Process Activation Model

In VHDL, processes can be suspended and resumed in a fairly robust manner using the wait statement to both suspend the process and dynamically define the criteria for reactivation (sensitivity list, timeout clause, and/or conditions). However, this presents two distinct problems for system level modeling. First, the dynamic characteristics (especially the ability to suspend a process from within a nested procedure call) make static analysis and scheduling of process networks difficult at best. Second, the suspension and activation characteristics of a process can only be defined by the process itself; there is no



Figure 2 - Multiplier Block

way to control these external to the process.

Consider the simple multiplier block shown in Figure 2. There are three alternatives for firing this block: whenever A or B changes, when both A and B have new available values, or on an external activation. Data flow systems typically implement the middle alternative or provide each option. To describe just the middle option in VHDL would require the following code.

```
process
  variable A_store, B_store : Real := 0.0;
  variable A_ready, B_ready : boolean := false;
  variable ready : boolean := false;
begin
  wait on A, B;
  if (A'Active) then
    A_store := A;
    A_ready := true;
  end if;
  if (B'Active) then
    B_store := B;
    B_ready := true;
  end if;
  if (A_ready and B_ready) then
    C <= A * B;
    A_ready := false;
    B_ready := false;
  end if;
end process;
```

While this VHDL would work, the performance is significantly reduced since the process is activated each time A or B changes value and requires five intermediate storage variables.

Design Refinement and Strong Typing

At the system level, designers are often concerned with the way data is moved from one task to another. At an abstract level, this data is not necessarily structured into its final form. A simple example of this is in DSP algorithm development where the designer will develop the algorithm using a floating point representation and, later, convert the data representation to a fixed point implementation. To handle this, system design environments usually separate the transfer object (token) from the data to which the token refers. However, in VHDL the transfer object (signal) is intimately tied into its data value due to strong typing and the signal assignment mechanisms.

As a simple example, consider a simple block which inserts a delay into a communication between two blocks.

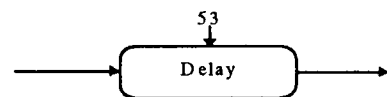


Figure 3 - Delay Block

Figure 3 shows a graphical representation of such a block. In most block-diagram-based system level environments, such a block is defined once in the library and used regardless of the data type of the communication. However, in VHDL a designer would need to define a different entity for each data type for which the block would be used.

Behavior, Communication and Timing

System level descriptions are a collection of communicating processes. To understand the timing

characteristics of the system, the designer needs to model the timing requirements of each task based on the requirements and constraints of the system. This usually means assigning task activation criteria, hard and soft task completion constraints, task prioritization and task preemption mechanisms.

As was noted above, process suspension/activation, communication, timing and, therefore, synchronization are defined within a process itself rather than external to a process. In order to effectively implement this over the complete process network of a system requires each process to adhere to specific synchronization semantics and timing behavior. This also requires each process to individually manage a potentially large number of global handshaking signals.

By including so much of the interprocess interaction semantics within a given process, reuse becomes virtually impossible except for the most simple cases.

Integrated Environments vs. Languages

The trend in both hardware and software is toward component-based design, where complex systems are constructed using a combination of predefined and custom components. On the surface, VHDL seems well-suited to this model. However, as a number of the above examples point out, it is virtually impossible to control VHDL components external to the component itself. This makes block-based design using VHDL, impossible without significant restrictions on the way blocks can be used.

In addition, system verification environments are inherently about stimulating a system description and analyzing the system's response. System level data is often complex ranging from audio/video signals to protocol packets to models of noise and stochastic effects. The analysis of such data is conducive to visual aids and processing. VHDL is not well-suited to integration with graphical analysis environments or with external interaction in general. Most VHDL systems compensate for this by providing external tools for collecting and analyzing data during simulation. However, with block-based design, blocks will require built-in debug and analysis capabilities.

Conclusions

The discussion in this paper has significant implications for HDLs and their role in the overall design and verification process. As we have seen, each design phase or level requires multiple models and robust verification strategies. What we have shown is that the analysis and verification methodologies used at each level are clustered around a collection of design representations most natural

for that level. Furthermore, each level addresses issues a concerns that are distinct and largely disjoint. Thus representing designs, a layered modeling and verification approach seems most appropriate.

In general, effective design and verification methodology for system-on-a-chip design are not HDL-based; HDLs can be used to represent certain aspects of a design within overall flow but they are not the primary, or even predominant, component of an overall design representation.

VHDL is a strong language which has enjoyed moderate success in adoption for electronic system implementation for boards, ASICs and FPGAs. VHDL, in its current form does not allow intuitive modeling or efficient tool development at the system level. The issues here go back to the fundamental foundations of the language (i.e. concurrent process model, strong typing, signal semantics etc.) and, therefore, there is no way to "repair" VHDL at this level.

Rather than trying to evolve VHDL to be a system level design language, we recommend that focus be placed on ensuring the future validity of VHDL for detail architectural and implementation modeling. This means focusing on requirements for cycle-accurate behavior, separation of timing and function, amenability for formal equivalence checking and support for integrating other domains of computation (like data flow, high-level discrete-event, mixed-signal, etc.).

As for system level design, we don't believe there is an alternative language looming on the horizon. Rather, integrated environments which allow modeling domains for specific aspects of a system will likely predominate.

Acknowledgments

The author would like to acknowledge the insightful suggestions and general contributions of Simon Curran, Stuart Swan, Oz Levia, Stan Krolikoski, Jim Rows, Misha Burich, Victor Berman and Patrick Scaglia of Cadence Design Systems.

References

- VSI Alliance (1996) Virtual Socket(TM) Interface proposal. Revision 0.9.
- Schulz, Steven E. (1996) Presentations, Results and Next Steps. System Level Design Workshop, Dallas, TX, USA. <http://www.cfi.org/sld/results/index.html>
- Lee, Edward A. and Alberto Sangiovanni-Vincentelli, "Comparing Models of Computation", Proceedings of ICCAD, San Jose, CA, Nov. 1995.
- J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: Framework for Simulating and Prototyping Heterogeneous Systems," Int. Journal of Computer Simulation, special issue "Simulation Software Development," vol. 4, pp. 155-182, Apr. 1994.