The Role of VHDL in the MCC CAD System

Ramón D. Acosta
Mark Alexandre
Gary Imken
Bill Read

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78759

Abstract

The VHSIC Hardware Description Language (VHDL), currently undergoing standardization by the IEEE, supports the hierarchical design, documentation, and simulation of a wide range of digital system abstractions. This paper describes a suite of utilities for manipulating VHDL designs that has been developed and integrated into the CAD System of the Microelectronics and Computer Technology Corporation (MCC). The MCC CAD System is a tightly integrated environment supporting the sharing of design information between heterogeneous tools via an underlying knowledge base built on top of an object-oriented distributed database. The VHDL utilities include an editing mode to provide syntactic assistance for writing VHDL, an analyzer to produce intermediate representations, a compiler to translate the intermediate representations into directly executable LISP functions, an elaborator for generating simulation models from complete designs, and a simulator for these models. Experimentation, continued development, and several important extensions to the CAD System VHDL utilities are in progress.

1. Introduction

The VHSIC Hardware Description Language (VHDL) Version 7.2 was defined in 1985 under government contract as a common hardware description language for companies developing government-sponsored hardware designs [Design & Test 86; Shahdad, et al. 85; VHDL 85]. More recently, the IEEE has undertaken an effort to promote VHDL as an industry-wide standard, using Version 7.2 as a baseline. A draft of the VHDL standard proposed by the IEEE, which has evolved substantially, has been released [VHDL 87], and final approval of an IEEE standard for VHDL was granted in December, 1987.

VHDL is intended to support the hierarchical design, documentation, and simulation of hardware from simple logic gates to complex digital systems. Hardware designs can be specified as purely behavioral descriptions, data-flow descriptions, structural descriptions, or any mixture of behavior, data-flow, and structure. VHDL supports design specification using both top-down and bottom-up approaches. The language shows strong influence from Ada[1] [Ada 83] both in syntax and semantics, particularly in the areas of data types and abstraction mechanisms. The versatility of VHDL allows it to be used for a variety of different circuit technologies while remaining independent of underlying implementations.

The generality of VHDL and its expected growth as an industry standard has led the VLSI CAD Program of the Microelectronics and Computer Technology Corporation (MCC) to integrate VHDL into its CAD System. A suite of utilities has been developed for editing VHDL designs, analyzing VHDL designs to produce and store intermediate representations, compiling portions of intermediate representations into directly executable form, elaborating completely configured hierarchical designs, and simulating the resultant models. Originally these utilities were based on VHDL Version 7.2, but recently they have been converted to support a subset of the IEEE draft standard (1076/B), with an ongoing effort to extend that support to cover the full IEEE standard for the language.

Various VHDL design systems, most notably those of IBM [Saunders 87], Intermetrics, and CAD Language Systems, Inc., have been implemented to support analysis, storage, and simulation of VHDL designs. The utilities described in this paper primarily differ from previous approaches in the following aspects:

- Integration into a large CAD system with a distributed knowledge base
- Adoption of the IEEE standard for VHDL
- Usage of Common LISP [Steele 84] as the implementation and the target simulation language
- Heterogeneous simulation using VHDL and other MCC CAD System representations

While our implementation adheres rigidly to the syntax and semantics of VHDL as laid out in the Language Reference Manual [VHDL 87], the MCC CAD System does not implement the proposed VHDL Support Environment. The language is used only as a notation for formal and executable specification of digital designs within the context of the MCC CAD System's integrated design methodology, such that considerable flexibility in using the VHDL utili-

[1] Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

ties in conjunction with other CAD System tools is readily available to the hardware designer.

The rest of this paper is structured as follows. Section 2 gives a general overview of the MCC CAD System to provide a framework for discussing the integration of VHDL. A brief summary of the major features of VHDL itself is given in Section 3 with a view to presenting what may be unfamiliar terms. Section 4 describes each of the VHDL utilities in turn, as well as the flow of data and control between them. Editing, analysis, and simulation of VHDL within the CAD System are described in Sections 5 through 7. Section 8 describes how the VHDL utilities support stand-alone usage, independent of the MCC CAD System. Finally, Sections 9 and 10 present some future directions and conclusions.

2. The MCC CAD System

The implementation of VHDL is one aspect of the MCC CAD System, currently under development by the MCC VLSI CAD Program. The MCC CAD System encompasses a large, multi-faceted effort aimed at building a tightly integrated, hierarchical CAD system capable of supporting chip and system-level design for VLSI technologies (see Figure 1). Key features of the system include the following:

- Tight integration between different tools
- Interactive design environment through a common user interface
- Extensive use of graphics
- Common data structures that support expressiveness (for knowledge-intensive tools), and efficiency and performance (for computationally-intensive tools)
- System-wide use of hierarchical design modules and reusable design knowledge

A variety of tools are being built for CAD activities such as editing, simulation, timing analysis, verification, test generation, and layout. The current computational environment is a distributed network of interactive LISP Machine workstations and mainframes running UNIX[2] (for compute-intensive work).

An important component of the CAD System is the Design Objects Storage Subsystem (DOSS) [Weiss, et al. 86], a distributed object-oriented database that serves as a repository for data common to all tools throughout the system. Most design information stored in DOSS is structured as part of the CAD Knowledge Base, a semantic network implemented in MCC's CAD Design Knowledge Representation System (CADRES). CADRES is a frame-based language modeled after the KL-ONE knowledge representation system [Brachman, Schmolze 85]. The CAD Knowledge Base is organized through CADRES into a taxonomy of concepts according to a well-defined semantics for hierarchical attribute inheritance. Each concept is represented as a frame with its own attributes and links to other concepts within the knowledge base. Concepts can be added, accessed, modified, and deleted uniformly by different tools within the CAD System. CADRES concepts provide considerable flexibility to individual CAD tools for the structuring and sharing of data in the Knowledge Base.

As discussed below, the MCC CAD implementation of VHDL makes extensive use of the Knowledge Base to store and retrieve analyzed VHDL designs and to access design information from other CAD tools.

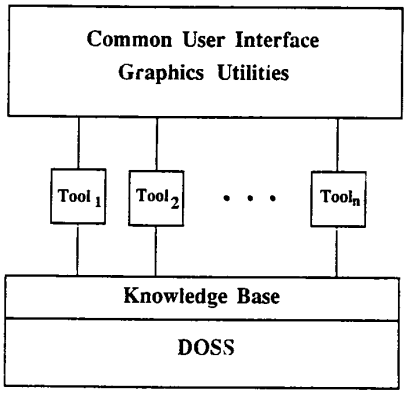

Figure 1. MCC CAD System Architecture.

3. VHDL Language Overview

VHDL is a precise notation for hierarchically describing the functionality and organization of hardware systems at various levels of abstraction. A *design entity* in VHDL is defined by an *entity declaration* and an associated *architecture body*. The *entity declaration* defines the interface of a design entity in terms of its externally visible ports and generic parameters. Several design entities may share the same entity declaration in order to describe different views of the same component or different components with the same interface. The *architecture body* specifies the input/output transformations or the internal organization of a design entity using any combination of structural, data flow, and behavioral design styles. Statements within an architecture body that correspond to these three styles, respectively, include:

- *component instantiation statements* for specifying the substructure of a design entity by the interconnection of its components
- *concurrent signal assignment statements* for evaluating arbitrary expressions and assigning the resultant waveforms to signals
- *process statements* for grouping together sequentially executed statements

Declarations for *subprograms* (*procedures* and *functions*) and *objects* (*variables*, *constants*, and *signals*) are allowed in a number of VHDL constructs, giving designers considerable latitude in scoping and visibility. Sophisticated schemes for defining scalar and composite abstract data types and subtypes, much like those available in Ada, are provided.

[2] UNIX is a trademark of AT&T Bell Laboratories.

VHDL also includes an encapsulating mechanism inherited from Ada, the *package*. Packages allow the sharing of data types, objects, and subprograms independent of the normal hierarchy of nested declarations. They also provide information hiding for better control of modularity.

VHDL also includes a higher-level encapsulation for design entities and packages, the *library*. Libraries establish separate namespaces in which designers can create and share VHDL design entities and packages.

4. The Implementation of VHDL

The MCC VLSI CAD Program is currently engaged in building a number of utilities for VHDL. These utilities are presently implemented in Common LISP and run on the Symbolics[3] LISP Machine as part of the CAD System, sharing access across a network to a common database managed by DOSS. The VHDL utilities also can be run in "stand-alone" mode, that is, without the rest of the MCC CAD System (see Section 8).

Our implementation of VHDL can be conceptually divided into five major utilities:

- An Editing mode for VHDL Entities
- A VHDL Analyzer
- A Compiler into LISP for VHDL Entities
- An Elaborator of VHDL Entities for Simulation
- A Simulation Interpreter Engine for VHDL Entities

A detailed diagram showing data-flow connections between the VHDL utilities and the CAD System support facilities appears in Figure 2. We now proceed to describe the VHDL utilities and their relationship to each other according to the illustration in Figure 2.

4.1. Editing mode for VHDL Entities

The Editing mode for VHDL Entities (**EVE**) is an enhancement of the native ZMACS editor on the Symbolics LISP Machine to support easier entry and modification of VHDL text. It is implemented as a new so-called "major mode" for VHDL under ZMACS, similar to the modes for LISP and other languages. EVE supplies automatic code indentation and permits display and insertion of syntax templates for all VHDL language constructs.

The goal of EVE is to facilitate the task of learning what will be for many designers a completely new language and to encourage a standard formatting style that promotes easier human understanding of VHDL designs. Nevertheless, the assistance provided by EVE is entirely optional and non-intrusive, and may therefore just as easily be ignored.

4.2. VHDL Analyzer

The VHDL Analyzer (**VANA**) accepts the source text for VHDL designs and checks them for syntactic and semantic validity against the language definition. In so doing, it builds an intermediate representation of the source in CADRES, which forms the basis for later manipulation of VHDL designs by other utilities. For each entity in an analyzed VHDL design, from libraries and packages, to entity declarations and architecture bodies, to blocks, processes and components, as well as signals, variables, and their data types, VANA creates a unique CADRES concept. These new concepts representing a VHDL design are then saved within the Knowledge Base, where they can be shared with other designers using the CAD System.

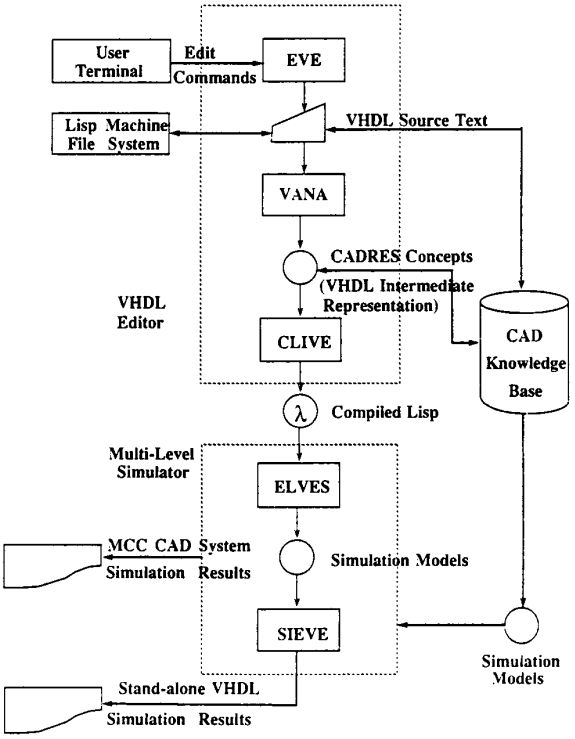

Figure 2. VHDL Utilities Data-flow Diagram.

4.3. Compiler into LISP for VHDL Entities

The Compiler into LISP for VHDL Entities (**CLIVE**) takes previously analyzed representations of VHDL produced by VANA and translates those parts containing only sequential statements into equivalent LISP expressions, which it then compiles and installs as functions in the LISP execution environment. These compiled LISP functions are then available during simulation of any design model containing components represented by the VHDL source, at which time they allow for considerably faster execution than would be the case for a purely interpretive model.

Sequential statements handled by CLIVE are found in VHDL processes and VHDL subprograms (procedures and functions). CLIVE also compiles those concurrent statements that are the semantic equivalents of a process, such as the concurrent signal assignment statement. In summary, CLIVE compiles those portions of a VHDL design that would roughly be considered behavioral rather than structural.

[3] Symbolics is a trademark of Symbolics, Inc.

VANA automatically calls CLIVE each time a design unit, such as a package or architecture body, is successfully analyzed. CLIVE may also be called by the ELVES, which is described below.

4.4. Elaborator of VHDL Entities for Simulation

The Elaborator of VHDL Entities for Simulation (ELVES) sets up a model for a completely configured, hierarchical VHDL design in preparation for simulation of that design. ELVES handles the elaboration of all VHDL entities, including the initialization of statically allocated signals and the creation of individually executing, autonomous VHDL processes. ELVES also expands any VHDL generate statements, which allow VHDL structural designs to be parameterized [VHDL 87, Ch. 12].

The ELVES can call CLIVE as needed for any previously analyzed VHDL process or subprogram whose compiled representation is not already present on the local host machine. This can happen when, for instance, the design entity being elaborated was not analyzed on the local machine, but on some other machine in the network. It also might happen when the local machine has had to be rebooted since analysis, thereby clearing out the compiled functions originally generated. Since the results of VANA's analysis are saved in the Knowledge Base for anyone to retrieve, a potential point of redundancy has been eliminated. Thus, it is never necessary for cooperating designers to reanalyze shared designs already in the Knowledge Base.

4.5. Simulation Interpreter Engine for VHDL Entities

The Simulation Interpreter Engine for VHDL Entities (SIEVE) handles all aspects of the VHDL simulation cycle, including signal assignment and propagation, process execution and suspension, and dynamic type checking. The SIEVE uses both the model from the ELVES and the functions compiled by CLIVE to run a VHDL simulation in a combination of interpretation and direct execution. The SIEVE can also manage its own time queue for event scheduling, but this is not typically necessary in its normal embedded configuration within the CAD System.

5. The VHDL Editor Tool

In their primary function as part of the MCC CAD System, EVE, VANA, CLIVE, ELVES and the SIEVE are integrated into a unified environment of CAD tools. This environment is layered on top of a network of LISP Machines and mainframes operating under UNIX that share a centralized database. Again, tight integration and data sharing between tools is the most prevalent design theme in the CAD System architecture. The VHDL design utilities observe and take advantage of this methodology.

As part of the MCC CAD System, the EVE, VANA and CLIVE utilities are integrated into a single tool called simply the VHDL Editor. The VHDL Editor can be used to interface with the CAD Knowledge Base, where both VHDL source text and analyzed representations of VHDL designs are permanently stored and shared with other designers; to edit new or existing VHDL descriptions; to invoke analysis of those descriptions; and to translate portions of VHDL designs into equivalent compiled LISP functions. These operations are either implicit or controlled by the user through a few simple options selectable from within the VHDL Editor.

Previously written VHDL designs obtained from external sources may also be entered into the MCC CAD System from text files. Similarly, VHDL designs composed within the VHDL Editor can be saved to text files for export to other systems. Structural descriptions of designs entered through other editors in the system can also be converted to VHDL text files for export to other systems.

6. VHDL Design Representation in CADRES

The existence of the Knowledge Base in the MCC CAD System is motivated by the desire to have knowledge-rich database facilities for CAD tools that support the extensive reuse of knowledge in the design cycle. By using VANA to build and store CADRES concepts in the Knowledge Base, the VHDL utilities take advantage of the system's knowledge structuring capabilities, while at the same time allowing other CAD tools to access and manipulate VHDL design information.

The modularity features of VHDL are straightforwardly modeled with CADRES. VHDL libraries become concepts that are linked to the concepts for the library units they contain, such as packages or entity declarations. A package concept is, in turn, linked to the concepts for the objects, data types, and subprograms that it defines. The concept for an entity declaration is linked to the concepts for all of its associated architecture bodies. With the results of VHDL analysis stored in this manner, the Knowledge Base obviates any need for a separate VHDL storage facility.

In addition, this approach allows leverage off other knowledge-based tools, such as the Concept Editor, a tool that enables designers to browse in a structured manner through the taxonomy of concepts representing all uninstantiated design knowledge in the system. In this way, designers can retrieve subprograms and designs from previous work that may be usable in the context of their current work. This complements the notions of hierarchy and modularity supported by VHDL.

Lower-level syntactic and semantic aspects of VHDL are also modeled quite naturally with CADRES concepts. For example, the structured inheritance mechanism between concepts can be modified to handle identifier scoping between nested declarative regions in VHDL. VHDL's strong Ada-like facilities for user-defined data types, where subtypes are derived from broader base types, are directly representable as a taxonomy based on inheritance of the set of values and applicable operations for each type. Furthermore, the many defaults for unspecified values and attributes in a language as large as VHDL can simply be propagated through the hierarchy of concepts.

The ELVES and SIEVE also make use of the CADRES representation of VHDL. The simulation model

built by the ELVES is based upon the network of concepts constructed by VANA and retains numerous pointers back into that network so that diagnostics and error messages can be given to the user during simulation in terms that are meaningful with respect to the original VHDL text. Concepts representing VHDL data types are used in the SIEVE for the elaboration of new subtypes and for dynamic type checking, such as is required when a value is assigned to an object.

CADRES concepts will eventually form the basis for a tool that combines editing with concurrent syntactic and semantic analysis of VHDL designs. This tool is briefly described in Section 9.2.

7. VHDL Simulation and the Multi-Level Simulator

The ELVES and SIEVE utilities are not part of the VHDL Editor tool. Rather, they are invisibly embedded in a tool known as the Multi-Level Simulator, which controls the simulation of all designs composed within the MCC CAD System. Designs written in VHDL may be freely embedded within larger design contexts created using other tools in the CAD System, such as interactive schematic-capture tools. When such heterogeneous designs are configured for interactive simulation, the Multi-Level Simulator automatically calls the ELVES as VHDL designs are encountered. Likewise, when these designs are executed in the Multi-Level Simulator, control passes automatically between the Multi-Level Simulator and the SIEVE as signals are propagated into and out of components described in VHDL.

The Multi-Level Simulator uses an event-driven simulation algorithm for logic and switch-level simulation. The SIEVE is implicitly activated only as needed under control of the Multi-Level Simulator. They share a common time queue for events that is managed by the Multi-Level Simulator, which accepts scheduling requests from the SIEVE for signals and processes internal to the VHDL components. The user interfaces only with the Multi-Level Simulator and its associated tools, including the Hierarchy Controller (a tool for design configuration management) and the Waveform Editor (an input stimulus and output display tool). Thus, the Multi-Level Simulator and the SIEVE can be said to stand in a "master-slave" relationship within the environment of the MCC CAD System.

This complete integration of the SIEVE with the Multi-Level Simulator enables mixed designs of components specified with VHDL and components created with other CAD tools to be simulated. Furthermore, use of VHDL with the Multi-Level Simulator allows effective efficient solutions to two major problems in the simulation of large designs.

First, the creation of input stimuli for the simulator is time-consuming and usually requires the designer to learn a different specification language. In the MCC CAD System, VHDL is used as the input stimuli language so that the designer needs to learn only one language for describing his design and the stimuli. This also allows the designer to simulate and debug the input stimuli without requiring simulation of the entire design.

The second major problem solved with this approach is that designers can create dynamic monitors for the simulator. Monitors that conditionally report state or error conditions during the simulation can be easily built by the designer in the same language that describes the design. An input stimuli/monitor can be created to provide stimuli for the simulation that can be modified based on state or error conditions that occur during simulation. The description of input stimuli and monitor modules as VHDL design entities allows them to be processed just like any other design entities.

8. VHDL Outside the MCC CAD System

When run in stand-alone mode, the VHDL utilities are individually accessible as LISP functions in the world of the LISP Machine. EVE runs automatically as a major mode in any ZMACS editor window. VANA can be invoked with a function that takes input from any VHDL source text file. The analyzed representation in this case, of course, is saved only on the local machine. As in the full CAD System, CLIVE is invoked automatically by VANA. The ELVES can then be called via a LISP function to create a model for simulation. An input pattern application function initializes the input stimulus as desired, after which the SIEVE can be called for individual simulation cycles or a fixed interval of time steps.

This mode of execution for VHDL is primarily used as a detailed debugging technique within the MCC CAD Program. It is useful for isolating the VHDL utilities from other MCC CAD tools and allowing direct access to the basic LISP Machine but requires a correspondingly greater sophistication on the part of the user. It is expected that end users (designers) would only use the VHDL utilities within the much more user-friendly context of the entire MCC CAD System.

9. Future Directions

Ongoing work involving a source-level debugging environment, further integration of the VHDL utilities with other CAD System tools, and a language-based editor/compiler is outlined in this section.

9.1. Extended Simulation and Analysis Capabilities

The simulation of VHDL will be extended to provide a source-level debugging environment for designers. This is analogous to source-level debuggers that are currently available to software designers. The design of very large system and circuit designs is becoming more like a software design problem than the traditional hardware design problem. Thus, creation of this type of debugging environment for hardware designers will be required in the future.

The current use of VHDL in the MCC CAD System is restricted to simulation. In the future, use of an integrated intermediate representation in the form of CADRES concepts will enable analyzed VHDL designs to be submitted to performance analysis, timing analysis, and synthesis tools. A more seamless integration with schematic editing and layout tools is also expected.

9.2. A Language-Based Editor/Compiler for VHDL

The rather ambitious idea of **A** Language-Based Editor/Compiler (**ALEC**) for VHDL is to eliminate the distinction between editing, parsing, and syntactic/semantic analysis of programs or designs. We seek to systematize the development of ALEC for different languages as much as possible. CADRES concepts will play a key role in ALEC. Our approach is to treat the editing task as a direct manipulation of the underlying knowledge base. Nevertheless, we believe that a significant weakness of previous work in this area is the failure to maintain familiar text-oriented commands [Bruns 87; Reps, Teitelbaum 84], a mode with which the typical user is already comfortable. ALEC will strive to present itself as a normal text editor, with exceptional text manipulation and error detection capabilities available as required by the user.

For instance, in addition to automatic formatting and syntax templates as in EVE, ALEC will offer automatic completion of abbreviated names and commands (including those defined by the user), type-sensitive and context-sensitive selection menus, hypertext-like cross-references between declarations and usages, multiple program views and transformations, and immediate feedback on syntactic or semantic inconsistencies. When the user is finished editing in ALEC, the source text will be guaranteed to be free of errors, both in syntax and in semantics, and no further parsing or analysis will be necessary. This should allow the user to progress directly to subsequent phases of development without entering into the usual cycle of edit → compile → correct → recompile → ...

Work is under way to develop tools for the implementation of ALEC. The first application of these tools will be to create ALEC for VHDL. When completed, ALEC will supplant EVE and VANA as the foundation for a new VHDL Editor in the MCC CAD System.

10. Concluding Remarks

At present, the VHDL utilities for editing, analysis, and simulation available in the MCC CAD System can handle a large and significant subset of the IEEE draft standard [VHDL 87]. Designs involving more than 10,000 lines of VHDL have been analyzed and simulated, and performance improvements are continually being made. Our intention is to support the entire standard language. Experimentation and continued development is in progress towards the goal of achieving a seamless integration between the CAD System environment and the VHDL utilities. This should provide a strong focus for future work, including the use CADRES and the CAD System Knowledge Base as the foundation for building ALEC and corresponding language-based editors.

11. Acknowledgments

The authors would like to acknowledge the contributions of the following members of the MCC CAD Program to the development of the VHDL utilities and other parts of the CAD System discussed in this paper: Manoj Hardikar, Dave Newton, David Franke, Richard Von Blucher, and Steve Smith. Our thanks go also to Tim Winter, John Kuban and numerous others for patiently exercising the VHDL tools in an everchanging environment, to Kellye Sheehan for her thoughtful comments on this paper, and to Virginia Waters for the figures.

12. References

[Ada 83]
"Reference Manual for the ADA Programming Language," ANSI/MIL-STD-1815A-1983, United States Department of Defense, Springer-Verlag, New York, NY, 1983.

[Brachman, Schmolze 85]
R.J. Brachman and J.G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, Vol. 9, 1985, pp. 171-216.

[Bruns 87]
G.R. Bruns, "Generating an Editor for a Software Design Language," MCC Technical Report No. STP-325-87, Microelectronics and Computer Technology Corporation, Austin, TX, October 1987.

[Design & Test 86]
IEEE Design & Test of Computers, Special Issue on VHDL: The VHSIC Hardware Description Language, Vol. 3, No. 2, April 1986.

[Reps, Teitelbaum 84]
T. Reps and T. Teitelbaum, "The Synthesizer Generator," *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, 1984, pp. 42-48.

[Saunders 87]
L.F. Saunders, "The IBM VHDL Design System," *Proceedings of the 24th Design Automation Conference*, 1987, pp. 484-490.

[Shahdad, et al. 85]
Moe Shahdad, Roger Lipsett, Erich Marschner, Kellye Sheehan, Howard Cohen, Ron Waxman, and Dave Ackley, "VHSIC Hardware Description Language," *Computer*, Vol. 18, No. 2, February 1985, pp. 94-103.

[Steele 84]
G.L. Steele, *Common LISP: The Language*, Digital Press, Bedford, MA, 1984.

[VHDL 85]
"VHDL Language Reference Manual, Version 7.2," Technical Report No. IR-MD-045-2, Intermetrics, Inc., Bethesda, MD, August 1985.

[VHDL 87]
"VHDL Language Reference Manual," VHDL Analysis and Standardization Group (VASG) Draft Standard 1076/B, CAD Language Systems, Inc., Rockville, MD, May 1987.

[Weiss, et al. 86]
S. Weiss, K. Rotzell, T. Rhyne, and A. Goldfein, "DOSS: A Storage System for Design Data," *Proceedings of the 23rd Design Automation Conference*, 1986, pp. 41-47.