Accepted for publication in *IEEE Design and Test of Computers*.

SUAVE: Extending VHDL to Improve Support for Data Modeling[*]

Peter J. Ashenden
University of Adelaide

Philip A. Wilsey and Dale E. Martin
University of Cincinnati

Abstract

Designers are increasingly using VHDL for high-level modeling. However, their task is hindered by the lack of object-oriented and genericity features in the language. Experience in programming languages shows that these features significantly aid management of complexity in large designs and promote re-use of modules. SUAVE extends VHDL by adapting several object-oriented and genericity features from Ada-95. The extensions improve support for modeling in VHDL from system level down to gate level. This article describes the extensions and illustrates their use with examples.

Index Terms: VHDL, object-oriented, genericity, hardware modeling.

Introduction

VHDL [1, 6] is widely used by designers of digital systems for specification, simulation, and synthesis. Increasingly, designers are using VHDL at high levels of abstraction as part of the system-level design process. At this level of abstraction, designers describe the aggregate behavior of a system in a style similar to that of software. They model data in abstract form, rather than using any particular binary representation, and express functionality in terms of interacting processes that perform algorithms of varying complexity. They may defer determining which aspects of the modeled behavior are to be implemented as hardware subsystems, and which are to be implemented as software, until later in the design flow.

Designers using VHDL for high-level modeling can learn much from the experiences of the software engineering community. There, engineers have widely adopted object-oriented design and programming techniques for managing complexity through abstract data types (ADTs) and re-use. Programming languages include features for abstraction, encapsulation, inheritance, and genericity to support these techniques. Software engineers use the terms *object-based* to refer to a language that includes abstraction and encapsulation mechanisms, and *object-oriented* to refer to a language that additionally includes inheritance [10].

While VHDL is usable for modeling at the system level, it has some deficiencies that make the task more difficult than it need be. These difficulties center around language features (or lack of some features) for supporting complexity management. In particular, VHDL is currently somewhat less than object-based, as its encapsulation mechanism is weak; it is certainly not object-oriented, as it does not include any form of inheritance. While it does in-

---

[*] This work was partially supported by Wright Laboratory under USAF contract F33615-95-C-1638.

clude a mechanism for genericity, that mechanism is severely limited, allowing only parameterization of units by constant values. Thus, VHDL cannot readily be used to define secure, re-usable ADTs.

Our aim in the SUAVE (SAVANT and University of Adelaide VHDL Extensions) Project is to improve support for high-level modeling and re-use in VHDL. A number of previous proposals also address these goals (see sidebar). SUAVE extends the language with features for object-orientation and genericity, and generalizes some existing features. Extending VHDL in this way has the side-effect of improving its expressiveness at all levels of abstraction. In this article we describe our language extensions, argue for their need, and illustrate their use with examples. We describe the extensions in more detail in a separate report [3]. Most of the features that we have added to VHDL are adapted from Ada-95 [7] and are included largely for the same reasons that they are included in Ada-95 [4].

Design objectives

Our analysis of requirements for extending VHDL for system-level modeling [2] leads to the following design objectives:

1. to improve support for high-level behavioral modeling by improving encapsulation and information hiding capabilities and providing for hierarchies of abstraction,
2. to improve support for re-use and incremental development by allowing later bindings through type-genericity and dynamic polymorphism,
3. to preserve capabilities for synthesis and other forms of design analysis,
4. to support hardware/software co-design through improved integration with programming languages (e.g., Ada),
5. to support refinement of models through elaboration of components rather than through repartitioning, and
6. to