submitted to *VLSI Design*

PRINCIPLES FOR LANGUAGE EXTENSIONS TO VHDL TO SUPPORT HIGH-LEVEL MODELING*

Peter J. Ashenden
Lecturer
Dept. of Computer Science
The University of Adelaide
Adelaide, SA 5005
Australia

Philip A. Wilsey
Assistant Professor
Dept. of ECECS, PO Box 210030
University of Cincinnati
Cincinnati, OH 45221-0030
USA

ABSTRACT

This paper reviews proposals for extensions to VHDL to support high-level modeling and places them within a taxonomy that describes the modeling requirements they address. Many of the proposals focus on object-oriented extensions, whereas this paper argues that extension of VHDL to support high-level modeling requires a broader review. The paper presents a detailed discussion of issues to be considered in adding high-level modeling extensions to VHDL, including concurrency and communication, abstraction using entity interfaces, object-oriented data modeling, encapsulation, signal assignment semantics, shared variables, multiple inheritance, genericity and synthesis. Emphasis is placed on the importance of designing simple orthogonal semantic mechanisms that interact in well defined ways, and that integrate cleanly with exisiting language features.

Keywords: high-level modeling, VHDL, object-orientation, genericity, concurrency, communication.

1. INTRODUCTION

In recent years, as the complexity of hardware systems has increased, designers have been forced to include high-level modeling as a stage in the design flow. Specifying and simulating systems at a high level of abstraction allows more reliable capture of requirements and more extensive exploration of the design

---

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

space. High-level modeling and related activities have the potential to reduce development time and cost. However, in order to support high-level modeling, a design language must allow specification of data and behavior in an abstract manner [41]. It should not force the designer to make design decisions that are better deferred to later in the design flow. For example, it should not force the designer to choose between a hardware and a software implementation too early, or to specify a detailed communications protocol before partitioning the design.

Ideally, a design language should allow the designer to span the spectrum of abstraction from high-level down to implementation (invoking compilers or synthesis tools to realize the final implementation). It is desirable to a single design languages throughout the design flow, avoiding interface- and equivalence-checking problems that otherwise arise. VHDL [28], as it currently stands, is a hardware description language that is well-suited to modeling small- to medium-scale systems at levels of abstraction up to register-transfer level. However, it has some serious shortcomings when used for modeling large-scale systems or systems at a high level of abstraction. In order to remedy these shortcomings, complexity-management and abstraction techniques that have been used successfully in programming languages should be reviewed as candidates for extensions to VHDL.

An important development in the software engineering community is the movement towards object-orientation as a means of managing the complexity inherent in large software systems. Object-orientation allows a design space to be partitioned into manageable pieces with well-constrained interactions, and facilitates the reuse and evolution of modules. According to Booch, “object orientation involves the elements of *data abstraction, encapsulation,* and *inheritance with polymorphism*” in a language [8, page 181]. The ideas denoted by these terms are clearly defined and illustrated by Booch [7] and have be reviewed extensively by other authors, so we do not include any detailed discussion here.

Given the success of object-oriented techniques in the software domain, it is appropriate to consider their inclusion in a hardware description language such as VHDL. We expect that the use of object-oriented techniques in hardware description languages will help designers manage complexity, improve their productivity, and improve the reliability of the design process. However, object-oriented extensions should not be viewed as a panacea. Object-orientation focusses mainly on abstraction over data and the related operations, and does not address issues such as concurrency and communication. Successful extension of VHDL to support high-level modeling requires a broader review of these and other issues.

Our aim in this paper is not to present or support any particular extension to VHDL. Instead, we review previous proposals for extensions and attempt to categorize them to enable detailed analysis and discussion. Our categorization is based on the semantics of each proposal, insofar as the semantics are elaborated in each proposal. We attempt to identify some pitfalls that can trap unwary players and show that many of the previously proposed extensions encounter the pitfalls and violate principles of good language design.

When considering extensions to a hardware description language, it is necessary to identify the requirements of the language users, and to ensure that proposed extensions meet these requirements. Bergé *et al* [6] report on a survey of several European telecommunication companies and system houses to discover their requirements of a design language. In particular, the respondents were asked to identify their current problems using VHDL and their expectations for object-oriented VHDL. The responses reported were: