vVHDL: A Visual Hardware Description Language

D. L. Miller-Karlow and E. J. Golin
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801-2987 USA

Abstract

Complex hardware systems design demands the effective use of organized approaches to design. Recently, the VHSIC Hardware Description Language (VHDL) was developed for use in the design process. Traditionally, engineers have developed hardware descriptions based on schematic circuit diagrams, which are a visual notation. When using VHDL, designers are hampered by the cumbersome nature of the language syntax. We have developed a visual hardware description language, Visual VHDL (vVHDL), that incorporates the major features of VHDL. This paper presents the syntax of vVHDL and the design system used for developing vVHDL programs.

1 Introduction

Computer-aided design (CAD) tools revolutionized the design of electronic components by allowing convenient management of large amounts of data. Hardware description languages (HDLs) are an important component of CAD tools. An HDL description of a design is a precise representation that can be used for documenting, communicating, and simulating the design. Modern HDLs allow electronic designs to be described both structurally and behaviorally and thus provide an effective mechanism for developing designs as they evolve from abstraction to reality.

One difficulty in using HDLs to develop designs is the cumbersome nature of the HDL syntax. An HDL describes a complex, multifaceted system using a one-dimensional textual representation, which results in the obfuscation of many aspects of the design. One way to improve the usability of HDLs is to employ a visual language that represents HDL constructs graphically, rather than textually.

A visual approach to hardware design is very natural, since engineers have traditionally developed hardware descriptions based on schematic circuit diagrams, which are a visual notation. Although these diagrams are very useful for describing gate-level designs, complex systems also require modeling components in terms of behavior, to allow for simulation and testing. Our goal is to develop a visual language for hardware design that allows the graphical specification of both the structure and the behavior of hardware components.

We have developed a visual programming language based on the VHSIC Hardware Description Language [10] (VHDL), which may be thought of as providing a visual syntax for the VHDL language. The visual syntax makes it easier for engineers to develop designs by easing the burden of programming in VHDL, and by providing a uniform visual approach to creating models combining structure and behavior. The underlying computational model is VHDL, and the semantics of the visual language are defined by a translation into textual VHDL code.

This paper outlines the development of *Visual VHDL* (vVHDL) and describes its syntax and use. The next section contains a discussion of related work, and Section 3 gives a brief description of the basic VHDL language. Sections 4 and 5 present the vVHDL language and the visual design system constructed around it. Finally, current status and conclusions are given in Section 6.

2 Related Work

Visual VHDL draws on previous work in the development of visual languages. Several attempts have been made at graphical representations for hardware description languages. These include the graphical representation for STRICT [2] which uses structural diagrams and Petri nets to represent behavior, the use of statecharts to represent behavioral descriptions [3], and EXEL [4] which uses an interconnection of graphic icons to specify the behavioral synthesis process. A problem with these approaches is that they were developed to address specific problems and are not broadly applicable. A better solution is a user-friendly interface to a general purpose hardware description language.

Most of the work on VHDL has centered around providing an integrated programming support environment. One of the earliest attempts along these lines was the IBM VHDL Design System [1]. Although this system pulled together all of the necessary VHDL tools, little was done to alleviate the textual problems associated with VHDL. A second attempt, the VCOMP system [5], tried to reduce the burden of syntax constraints in VHDL by providing the programmer with a a set of menu choices that automatically generate code for common VHDL components.

The attempt which comes closest to vVHDL is the AFIT VHDL Environment [6]. Like the previous two, this system integrates a set of tools for developing VHDL programs. It also provides a graphical user-interface, called the Graphical VHDL User Interface (GVUI), which allows the user to enter VHDL behavioral code through the use of a typical schematic editor, and VHDL structural code as a series of interconnected, variable sized rectangles.

The goal of vVHDL is similar to that of GVUI, but vVHDL is based on visual programming, rather than just the use of a graphical interface. In vVHDL, shapes such as rectangles, arcs, and lines are used to represent the different components of textual VHDL. These shapes are then arranged according to a specific syntax. This is in contrast to the GVUI, in which shapes can be laid out in an arbitrary manner. In vVHDL the shapes *and* their spatial relationships describe a complete VHDL program.

Several visual languages have been developed based on existing textual languages that support concurrency, including Pigsty [7], pictorial Janus [8], and G-LOTOS [9]. The issues addressed by these languages are important to the development of vVHDL because modelling concurrency is a central issue in VHDL and the expectations of text-based VHDL will be transferred to vVHDL.

Pigsty is a visual language for concurrent Pascal. Pigsty uses a combination of text and graphics to represent a program which consists of one or more sequential processes. Boxes, representing processes, are connected by lines to show the communication links between them. Pictorial Janus (PJ) is a visual language for Janus, a concurrent constraint based language. PJ uses arrows and closed contours to depict behavior and is completely graphical. G-LOTOS is a visual language for LOTOS, a language for describing the interconnection of open distributed systems. Visually, G-LOTOS is the most complex of the three languages, employing a large number of different shapes for representing its parts. vVHDL incorporates aspects of all of three languages.

3 VHDL - The VHSIC Hardware Description Language

The VHSIC Hardware Description Language supports the development, verification, synthesis, and testing of hardware designs; the communication of hardware design data; and the maintenance, modification, and procurement of hardware [10]. VHDL allows the design of electronic components using a combination of structural, behavioral, and data flow models.

The basic building block of VHDL is the ***design unit*** which describes the design of a hardware component. A design unit consists of an entity declaration, which specifies the component interface, one or more architecture bodies containing implementations for the component, and an optional configuration statement. An entity may have several alternative architectures associated with it, allowing existing architectures to be replaced as the component design evolves.

```
architecture PROCESS_IMPL of CL is
begin
  process(X1, X2, X3)
    variable T1: BIT;
  begin
    T1 := X1 and X2;
    F <= T1 or X3 after TOTAL_DEL;
  end process;
end PROCESS_IMPL;
```

Figure 1: A VHDL Architecture Body

The architecture body specifies the function of the entity, and can be modeled in terms of structure, behavior or data flow. A structural architecture provides a hierarchical decomposition of the entity into subcomponents and interconnections, much like a traditional schematic. A behavioral model uses a procedural language to specify the behavior of a component. A data flow model consists of a

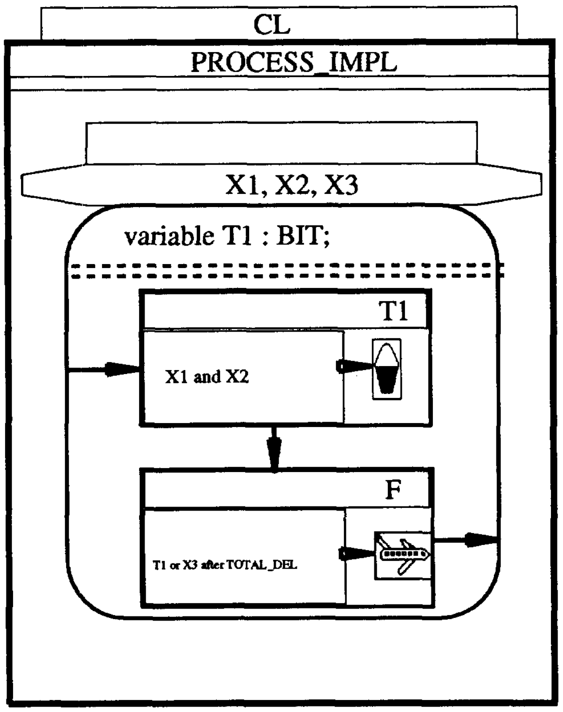


Figure 2: An Example of a VHDL program using the Shape and Flow Paradigm

collection of signal assignment statements which execute concurrently. Figure 1 gives an example of a VHDL architecture body consisting of a single process defined with data flow. The configuration statement specifies how several design entities are put together to form a complete unit. A VHDL hardware design is typically used as a basis for simulation, analysis or synthesis of a component, but can also be used for documenting or communicating a design.

4 A Visual Syntax for VHDL

Fitter writes that it is important for a diagrammatic language to both *redundantly record* the important information and to *reveal* the underlying process. [11] vVHDL achieves this by using similar shapes for similar VHDL constructs, and expressing sequential flow using arrows. We call this approach the *shape and flow paradigm*. Figure 2 shows an example of the shape and flow paradigm for the code in Figure 1.

The shape and flow paradigm represents encoded information in a relevant way by choosing shapes

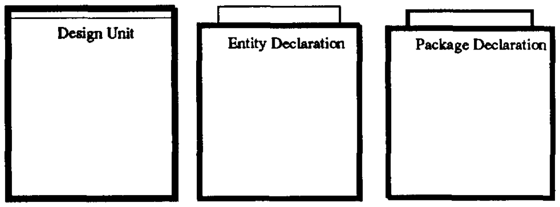


Figure 3: Declaration Units in vVHDL

and icons suggestive of the objects they represent. It redundantly records the important parts of the language by representing similar objects with similar shapes. The representation reveals the underlying process by representing concurrent and sequential statements with different shapes, and the prominence of the primitives reveals the hierarchical nature of VHDL constructs. Thus, the function of a vVHDL program is contained in its topology, and it is not necessary to refer to the textual details. The following sections describe the relationship between the vVHDL primitives.

4.1 Declarations and High Level Primitives

There are several declaration constructs in VHDL. The design unit is used for declaring a primary (i.e., top-level) unit. The entity declaration specifies the interface to a design and the package declaration gives the package interface. To emphasize their similarity, declaration units in vVHDL all have a thick rectangle as their main component as shown in Figure 3. These particular components will be referred to often to determine what the interface is to the piece of hardware being designed, and the prominent box makes them stand out so that they are easily located.

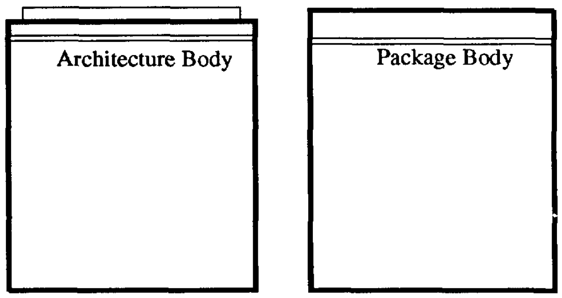


Figure 4: An Architecture Body and Package Body in vVHDL

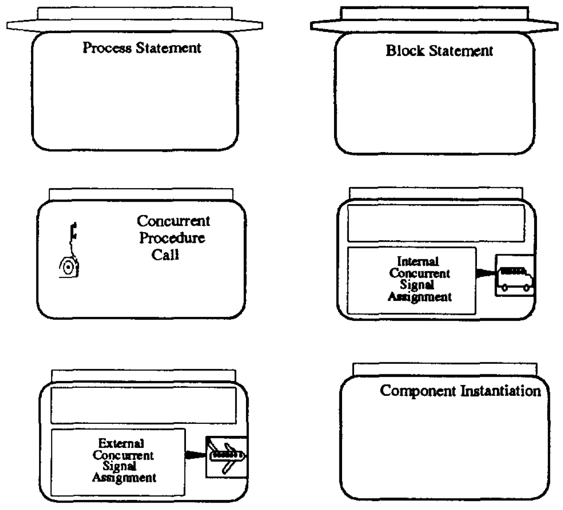

Process Statement
Block Statement
Concurrent Procedure Call
Internal Concurrent Signal Assignment
External Concurrent Signal Assignment
Component Instantiation

Figure 5: Concurrent Statements in vVHDL

The architecture body and the package body contain the implementation of a design entity (an architecture) or a package. Figure 4 shows these two constructs. Both the architecture body and the package body primitives are drawn with a thin rectangle as their dominant shape. They have shapes that are similar to, but less emphasized than their respective declaration units. By making them less conspicuous, the hierarchical relationship between the primitives is emphasized.

4.2 Statements

The contents of an architecture body consists of concurrent statements. Figure 5 shows the six types of concurrent statement primitives: the process statement, the block statement, the concurrent procedure call, the internal and external signal assignments, and the component instantiation. The dominant shape of these figures is a rectangle with rounded corners. The lines are thinner than the rectangles used in the architecture body and the package body to show that concurrent statements must be contained by architecture and package bodies. An optional label may be placed in the rectangle atop a concurrent statement.

Sequential statements are used to specify behavioral models. The sequential primitives are shown in Figure 6. To distinguish them from the concurrent statements, sequential statements have a rectangle as their dominant shape. This rectangle is the same

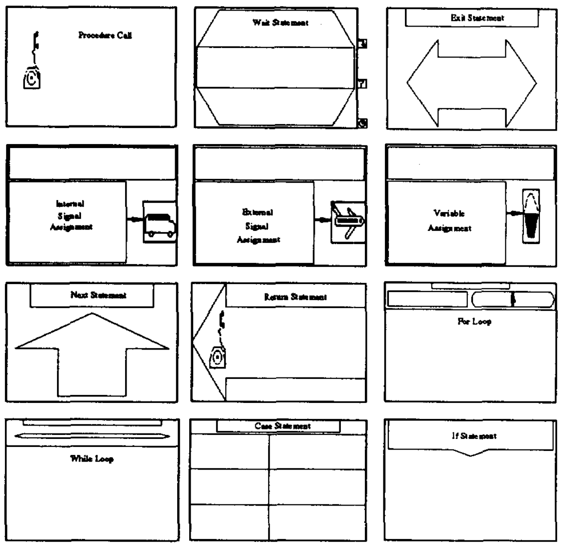


Figure 6: Sequential Statements in vVHDL

thickness as used in concurrent statements to show that sequential and concurrent statements have a similar hierarchical station.

The final set of similar primitives are the assignment statements. VHDL allows assignment to both internal and external signals and variables, and both sequential and concurrent signal assignments are permitted. Figure 7 shows the assignment statements in vVHDL. They all have three internal components: an upper rectangle, a lower rectangle and an icon. The upper rectangle holds the name of the signal or variable that is being assigned to, the lower rectangle holds that value that is being assigned, and the icon represents the type of the assignment. An internal signal assignment is represented by a bus icon, an external signal assignment is represented by an airplane icon, and a variable assignment is represented by a bucket icon. The shape of the external part of the assignment statement reveals whether it is a concurrent statement or a sequential statement.

4.3 A vVHDL Example

As an example, we give here a complete design, adapted from Armstrong [12], to show how the vVHDL pieces fit together. Figures 8, 10, and 9 show the entity declaration and two architecture bodies for a function, ONES_CNT, that counts the number of ones in a vector of length three. One of the architecture bodies is a behavioral description and the

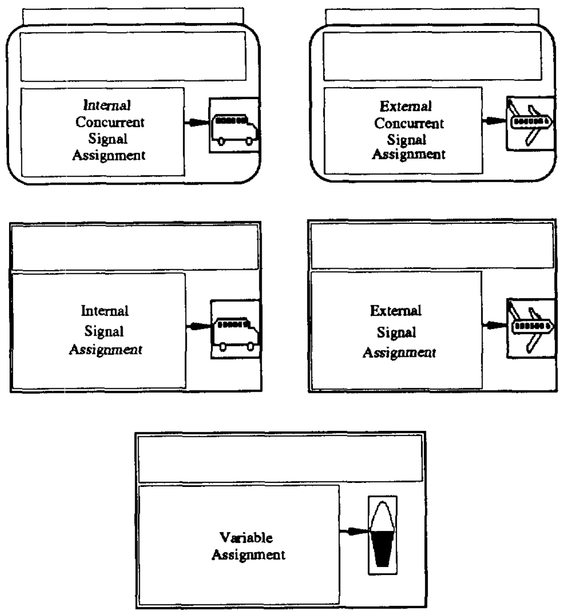
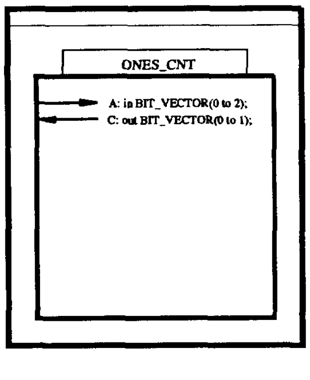

Internal Concurrent Signal Assignment
External Concurrent Signal Assignment
Internal Signal Assignment
External Signal Assignment
Variable Assignment

Figure 7: Assignment Statements in vVHDL

other is a structural description.

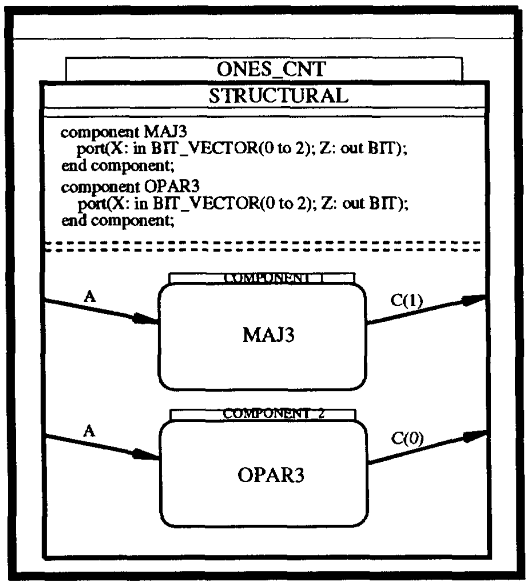
The ONES_CNT pictures show clearly which element is the entity declaration, which are the architecture bodies, and how they are related to each other. The entity declaration in Figure 8 displays the interface of ONES_CNT, and shows that it depends on one incoming signal, A, and one outgoing signal, C, along with the types of these signals.

Figure 10 contains an architecture body comprised of a concurrent statement, three sequential statements, and some variable assignments and external signal assignments. This design consists of a concurrent process which depends on the signal A. Inside of this process, a local variable, NUM is declared. When the value of A changes, NUM is initialized to 0. A for statement is then used to cycle through the vector A, testing each of its elements to see if they are equal to 1. If this condition is true, NUM is incremented. Finally, a case statement assigns a value to C based on the value of NUM.



```
A: in BIT_VECTOR(0 to 2);
C: out BIT_VECTOR(0 to 1);
```

Figure 8: Entity Declaration for ONES_CNT



```
component MAJ3
  port(X: in BIT_VECTOR(0 to 2); Z: out BIT);
end component;
component OPAR3
  port(X: in BIT_VECTOR(0 to 2); Z: out BIT);
end component;
```

Figure 9: Structural Description for ONES_CNT

Figure 9 gives an alternative architecture body for ONES_CNT. This architecture is a structural design consisting of two concurrent components, MAJ3 and OPAR3, that operate on the external signal A and whose output is a part of the signal C. It is not necessary to know anything about the code that describes the behavior of MAJ3 and OPAR3 to understand the structure of ONES_CNT.

5 The Visual Design System

We have constructed a *visual design environment* that allows an engineer to specify a component design by drawing a picture. The vVHDL environment generates textual VHDL code for the design which can be used with other tools for simulation or synthesis. The approach to building the environment is based on combining a graphical editor tailored to vVHDL with a compiler for the vVHDL language. The overall architecture of the system is shown in Figure 11.

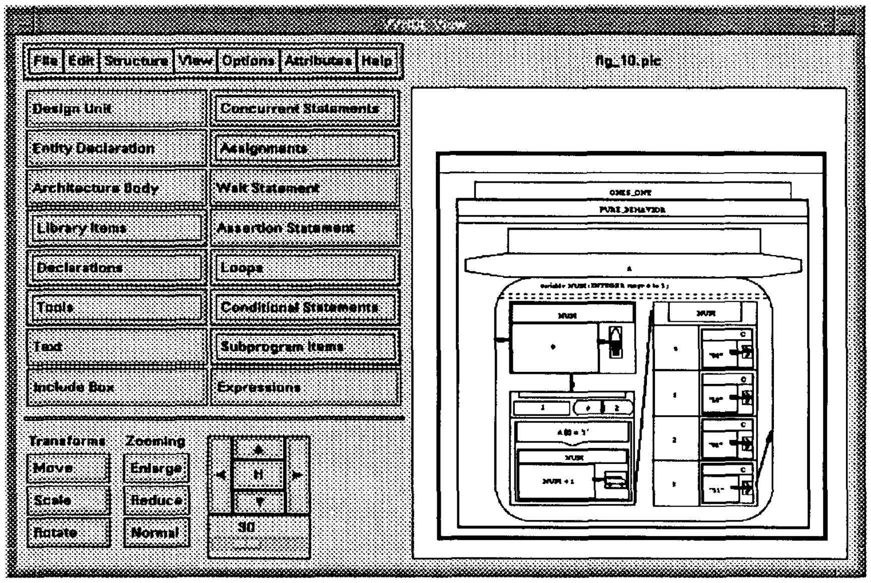


Figure 10: Behavioral Description for ONES_CNT, shown with vVHDL Editor

The vVHDL editor is constructed using the Palette system. [13] Palette is a *reconfigurable* editor for visual programs. It provides the underlying capabililites of a graphical editor, including the overall editor framework, automatic display facilities and simple editing functions. Palette also defines a framework for customizing an editor to a specific visual language. An editor for vVHDL, shown in Figure 10, was constructed by defining the graphical shapes that are found in the vVHDL language. The editor is also the user's interface to the vVHDL compiler. Selecting the `Parse` button causes a file to be created that contains the primitives from the current picture. This file becomes the input to the vVHDL compiler.

The vVHDL compiler is responsible for analyzing the input picture to determine the syntactic structure (i.e., parsing the picture) and then translating the picture into textual VHDL code. The visual syntax of the vVHDL language and its translation into textual VHDL is defined by an object-oriented picture layout grammar[14]. The graphical symbols and relationships are specified using C ++ classes and functions. The grammar is processed by the SPARgen spatial parser generator to produce a vVHDL compiler.

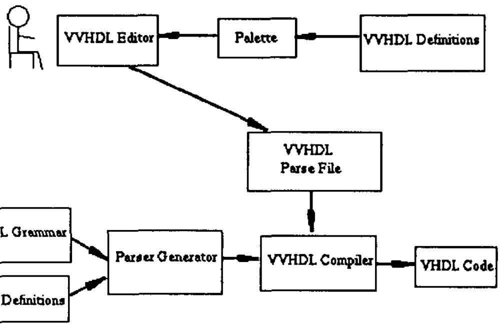


Figure 11: The vVHDL Design System

6 Summary

In this paper, an initial version of a visual language for VHDL has been presented. We have constructed an editor for vVHDL programs and a compiler which translates diagrams into textual VHDL code. Currently, the editor and language are being used by several designers developing electronic components for High Energy Physics data acquisition. [14] Feedback from these users is being used to refine the visual syntax and the editor functionality.

Presently, a sufficient subset of VHDL is supported by vVHDL to design components of arbitrary complexity. Some language features, such as declarations, are supported by including text fragments containing the VHDL code (a special construct is also provided which allows VHDL text to be inserted into a graphical design). Work on the vVHDL language and environment are continuing in several areas:

- We are extending vVHDL to cover the complete VHDL language rather than just a subset.
- We are developing visual mechanisms for features currently supported with text.
- We are developing alternative notations (e.g., state diagrams) for behavioral models.
- We are integrating the environment with an object-oriented database and object based message-passing integration framework.

Only time will tell whether vVHDL will be a success. The designers using the language agree that vVHDL has significant potential. It is easy to learn and appears to alleviate many of the syntax problems associated with VHDL. It improves understanding of the code's function by clearly distinguishing between sequential and concurrent processes. The current version provides an excellent foundation for visual hardware design.

Acknowledgements

The authors wish to thank Annette Feng for redrawing the figures and reviewing this paper. Mike Haney, Eric Hughes, George Tharakan and the other members of the SSC DAQE Design Environments project have contributed to the evolution of the vVHDL language. This research was supported in part by the UIUC Campus Research Board, the Texas National Research Laboratory Commission Project 47, and the National Science Foundation grant CCR-9108931.

References

[1] L. F. Saunders, "The IBM VHDL Design System," in *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 484–490, 1987.

[2] C. A. Kuszynski, T. Busfield, A. M. Koelmans, M. R. McLauchlan, and D.J. Kinniment, "Graphical Representation of a Hardware Description Language," *IEEE Proceedings on Computers and Digital Techniques*, vol. 137, pp. 462–467, November 1990.

[3] D. Drusinsky and D. Harel, "Using Statecharts for Hardware Description and Synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 8, pp. 798–807, July 1989.

[4] N. D. Dutt and D. D. Gajski, "Designer Controlled Behavioral Synthesis," in *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 754–757, 1989.

[5] P. R. Jordan and R. D. Williams, "VCOMP: A VHDL Composition System," in *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 750–753, 1989.

[6] Joe DeGroat, Kevin Berk, Douglas Pompilio, and Stephen Matechik, "The AFIT VHDL Environment," in *1988 Frontiers in Education Conference Proceedings*, pp. 324–329, 1988.

[7] M.-C. Pong, "A graphical language for concurrent programming," in *1986 IEEE Workshop on Visual Languages*, (Dallas, TX), pp. 26–33, June 1986.

[8] K. M. Kahn and V. A. Saraswat, "Complete visualizations of concurrent programs and their executions," in *1990 IEEE Workshop on Visual Languages*, (Skokie, IL), pp. 7–15, Oct. 1990.

[9] T. Bolognesi and D. Latella, "Techniques for the formal definition of the G-LOTOS syntax," in *1989 IEEE Workshop on Visual Languages*, (Rome), pp. 43–49, Oct. 1989.

[10] *VHDL Language Reference Manual (IEEE Standard 1076-1987).* New York, March 1988.

[11] M. Fitter and T. R. G. Green, "When do diagrams make good computer languages?," in *International Journal of Man-Machine Studies*, p-p. 236–261, 1979.

[12] J. R. Armstrong, *Chip-Level Modeling with VHDL*. Prentice-Hall, Inc., 1989.

[13] E. J. Golin, S. Danz, S. Larison, and D. Miller-Karlow, "Palette: an extensible visual editor," in *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pp. 1208–1216, Mar. 1992.

[14] E. J. Golin, M. J. Haney, E. Hughes, D. Miller-Karlow, and G. Tharakan, "Visual design with vVHDL," Tech. Rep. UIUCDCS-R-92-1745, University of Illinois, 1992.