VCOMP: A VHDL Composition System

Paul R. Jordan
Barron Associates, Inc.
Route 1, Box 159
Stanardsville, Virginia 22973

Ronald D. Williams
Center for Semicustom Integrated Systems
Department of Electrical Engineering
University of Virginia
Charlottesville, Virginia 22901

Abstract

An automated system for creating and managing VHDL design units is described in this paper. This system helps the hardware designer to learn the syntax of the VHSIC Hardware Description Language (VHDL) and to produce code which can be used for simulations. This demonstration prototype tool is written in C to run on a Sun workstation, and it produces IEEE standard 1076 VHDL code.

1. Introduction

The demonstration prototype system described in this paper is called VCOMP for VHDL Composition System. This system was written in C to run on a SUN workstation under the Suntools environment. The purpose of this work was to identify approaches to simplify the introduction of VHDL for hardware designers. A demonstration prototype tool was designed to permit testing of these approaches. The software system described in this paper is not a "production quality" tool, rather it is a vehicle for considering one facet of VHDL working environments. Several goals were established for this system:

- ⇒ It should serve as a front end for an existing commercial VHDL simulation system.
- ⇒ It should introduce VHDL to the hardware designer in a simplified form.
- ⇒ It should allow the designer to produce hierarchical VHDL model code quickly.
- ⇒ It should provide a simple method for preparing device tests and for running simulations.
- ⇒ It should be easily extended.

VCOMP offers several services to its users. It serves as an interface to a complex and powerful language (VHDL), and it permits the user to grasp the syntax of VHDL and to setup the complete design process for system simulation. It combines all of the elements of a simulation–based design environment in one convenient package for use on a high quality workstation. It uses the graphical capabilities of the workstation to provide improved representations of simulation input and output data. These enhancements to the design environment provided by VCOMP should simplify hardware description and testing.

2. The Need for VCOMP

The management and processing of large amounts of data is becoming a greater task for the electronics designer. The use of a description language, such as VHDL, enables the designer to represent a design efficiently and effectively to permit automated testing of its functionality before advancing to the production stage. However, the representation advantages offered by the richness and power of VHDL also can make it difficult to use by experienced hardware designers who are less experienced with VHDL. VCOMP is intended to help the designer who is first being introduced to the hardware description language. The assistance which VCOMP provides to produce simulation code could be a necessity for the novice VHDL user since errors in the design can be more readily found and corrected [1].

The design environment is automated around VHDL because of the need to produce models that can be transported [2]. Since VHDL has been adopted as an IEEE standard, this language provides an appropriate medium for the interchange of transportable models. Since it appears appropriate to capture designs in VHDL and it is also appropriate to reduce extraneous burdens on the designer, VCOMP was developed to explore the design support offered by an automated VHDL composition system.

3. VHDL: The Environment and Tools

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 [3]. The development of the language began in 1983 with personnel from Intermetrics, IBM, and Texas Instruments. The language became an IEEE Standard in December 1987. This section discusses several key elements in the syntax of VHDL and comments on the environment.

3.1. Description of VHDL

VHDL is a textual description of hardware at several design abstraction levels from the logic gate level to the system level [4]. The language is organized into design unit bodies as either a design entity, configuration, or package, with the internal code being declarations, specifications, expressions, and statements.

The primary abstraction in VHDL is the design entity, since it represents the inputs and outputs of an entire system, subsystem, board, chip, cell, gate, or other hardware abstraction with a well–defined function. The design entity consists of an entity declaration and an architecture body. The entity declaration defines the interface between the design entity and the environment, and the architecture body describes the internal function and organization of the design entity.

The architecture body is associated with a specific entity declaration to describe the body's behavior, dataflow, or structure. The statements in the architecture body are concurrent statements,

which execute asynchronously with no defined relative order. A configuration can also be used to describe the interconnection of design entities. This design body forms the desired path or connection to a specific entity.

Another VHDL abstraction is the package, used for defining common resources to be used in different design units. The package is a collection of declarations (types, subtypes, constants, attributes, functions, etc.), and any programs defining these declarations (functions, procedures). Packages are very similar to *include* files used when programming in C.

The description of VHDL offered here is necessarily very brief. However, the language is very rich. The reader desiring more details on the language is advised to consult the IEEE Standard VHDL Language Reference Manual [3].

3.2. VHDL Environment Tools

Several VHDL simulation environments have become available recently. Some of these create a full electronic development system around their simulators [5,6], while others just have a support environment for VHDL 1076 [7,8]. The VHDL environment used for this research is the Standard VHDL 1076 Support Environment [7].

The Intermetrics support environment used by this effort enables hardware designers to use VHDL efficiently for the storage of design units in a central design database through the use of several software components. One of these components is the analyzer which is a syntactic and static semantic checker of VHDL descriptions that translates correctly coded units into an Intermediate VHDL Attributed Notation (IVAN) for placement in the design database. A simulator then provides a means for checking the semantic correctness of a VHDL hardware module by constructing simulatable models from the IVAN code, executing the models, then generating reports from the simulation.

There have been many efforts to improve environments for supporting VHDL work. These efforts have ranged from VHDL syntax aids [8] to full environment CAD tools for easy entry and test of VHDL descriptions [6,5]. The system described here interfaces to both the source code and report sections of the Intermetrics environment, and VCOMP provides the functions of a tutorial, a design developer, and a VHDL composer that is easily used.

4. The VCOMP System

A CAD system to create a usable human interface to the VHDL environment was developed after many VHDL hierarchical models had been generated. The emphasis of VCOMP is to help the designer create simulatable VHDL code while presenting a user–friendly facility for the insertion and extraction of signal waveforms for testing a design. The following is a list of the features of VCOMP:

- ⇒ User–friendly system with mouse/keyboard entry of VHDL design: creation of VHDL units such as packages, entities, and architectures.
- ⇒ Multiple windows for interactive display of: VHDL source text, waveform signals for testing, and simulation signal reports.
- ⇒ Hierarchical design support of structural and behavioral modules

The VCOMP system was developed on a Sun workstation in a graphics–based environment to support the writing and simulating of 1076 VHDL code. VCOMP displays a large screen–size base frame when executed. Several subwindows exist within this base frame, and four are always visible. The menu subwindow offers the main selections of the system. The VHDL development subwindow provides a structured entry of items for forming desired units of VHDL code. The VHDL text subwindow is used to display VHDL code and files for reading and editing. The last visible window is used in creating and displaying waveform signals.

Entry of designs into the system is made through the use of a mouse and the keyboard. The mouse is used to select items and to position the cursor in the text subwindow when editing. The main menu has button selections with eight possibilities: *Help*, *Package*, *Entity*, *Architecture*, *TestBench*, *Draw*, *Simulate*, and *Exit*. The *Help* selection will display text which can aid the designer in using the system at the current level. Thus a main help guide is available for the overall system as well as help guides for each of the seven other selections. The order of operations within VCOMP is important. The designer must build an entity before the architecture, but a package can be created at any time before analyzing and simulating the VHDL code. A test bench, which is the top–most level in a system's hierarchy, must be built last when it uses user–defined modules. The simulation and draw portions will work only if the appropriate files exist before execution.

The *Package* portion allows the designer to place common declarations and bodies in one design unit which will be used by several other design units. Thus the user can create new types, subtypes, constants, attributes, and functions which are specific to the design. In building a package, VCOMP helps the designer with the syntax of these declarations and then allows editing of extra declarations and filling in of blanks in user–defined functions within the package body.

The *Entity* selection provides the designer with a quick method for correctly creating the interface to a design body. After giving the name of the entity, choices are available to enter a package, a generic, or a port to the entity. For these selections, the designer is prompted for names, types, or modes to create the desired code. For each entry, the text subwindow displays the new file, allowing the user to edit this file after all automatic additions have been made. Then the corresponding architecture can be created.

The *Architecture* selection first prompts the designer for the name of the architecture and its corresponding entity. Then a menu of choices is presented to allow the formation of the design body. These choices are to connect a component within the architecture, to include a package, or to add a signal assignment statement. The connection, or instantiation, of a component may be from either a pre–defined or user–defined component. This involves connecting the ports and generics from the component's entity. After all constructs and connections have been made, the user can edit the resulting architecture before exiting.

The *TestBench* portion is similar to the architecture selection, yet it contains more choice selections. The primary difference is in the entry of signals. In the test bench, the designer can create a clock process quickly by giving the signal name, type, and clock period, or a signal waveform can be constructed using the waveform subwindow. When adding waveform signals to the test bench, the designer uses the mouse to add transitions along a time axis by clicking the right mouse button. When several signal waveforms have been added, the user can switch between them by hitting the middle mouse button or the last transition of the current signal can be repeated by clicking the right mouse button. A display of the time selected is visible so that the designer can verify the time selection made. If the simulation end time differs from the maximum time given for the time axis, or if waveforms are not used, the designer can enter the simulation stop time as well.

The *Draw* selection permits the designer to give both an entity and architecture, in either separate files or the same file, and the system creates a subwindow which then calls a separate VHDL

schematic display program with the designated entity/architecture pair [9]. The VHDL text for the entity/architecture pair is shown in a text subwindow as the schematic display routine runs. The display program parses the 1076 VHDL code and produces a schematic over the VHDL text subwindow. This routine also allows the designer to move the schematic symbols to improve the layout and view the design in hierarchical form. Note that the schematic display program has resulted from a separate effort and is therefore not discussed further in this paper.

The *Simulate* section is an interface between the VHDL design units and the Intermetrics VHDL support environment. This selection permits the designer to create a signal report control file or a simulation batch file for use in the VAX/VMS operating system. Appropriate files and commands are exchanged between the VCOMP workstation and the VAX/VMS system. The signal report control file is used to retrieve signal reports from a system simulation. If a test bench is given, VCOMP will automatically parse the test bench file for signals, and include these signals in the report. The batch command file is automatically created after the designer gives the test bench file for simulation. Then VCOMP prompts the user for any lower–level components called in this test bench, and the entity and architecture names are entered. Both of these files are used together to run a complete simulation in the Intermetrics environment. After running the simulation, a report file of signal values (boolean, vector, or integer types) can be parsed and displayed in the waveform subwindow for better comparison and quicker analysis of the results.

5. VCOMP Example

A 4–bit parallel register constructed from the gate level is offered here as an example hierarchical design. The VHDL design units were developed using VCOMP and then transported from the Sun to a VAX/VMS for full simulation. The results were then returned to the Sun to be observed in the VCOMP simulation display system.

The circuit design for this register could be developed at several different levels using VHDL. The gate level hierarchy was chosen to illustrate the design methodology for VCOMP, even though it involves more VHDL code than would be required with a purely behavioral description. A parallel register was built of the following gate components: 2–input AND, 3–input NOR, and an Inverter. These standard components exist in a component library available to VCOMP, and the system is used to simplify the selection and interconnection of these library modules. The hierarchy of the register involves the representation at four levels. Initially an SR Flip–Flop was built. Then, a D Flip–Flop was formed. Next, the 4–bit parallel register was built. Finally, the test bench level was created to apply the test stimuli. VCOMP was used to build the entity and architecture bodies at each of these levels.

The test bench output code from VCOMP for the parallel register is shown in Figure 1. A portion of the report file and

```
entity TestBench is end TestBench;

library std;
use std.simulator_standard.all;

use Work.uva_pack.all;

architecture preg_test of TestBench is

   component preg_ent
      generic ( delay : delay_integer);
      port (X0, X1, X2, X3, CLK, CLR : in level;
         Z0, Z1, Z2, Z3 : out level);
   end component;

   for all:preg_ent
      use entity work.preg_ent (preg_arc);

   signal X0, X1, X2, X3 : level;
   signal CLK, CLR, RESET : level;
   signal Z0, Z1, Z2, Z3 : level;

begin

   CKT: preg_ent   generic map (1ns)
                      port map (X0, X1, X2, X3,
                         CLK, CLR, Z0, Z1, Z2, Z3);

   X0 <= '0' after 10ns,
            '1' after 40ns,
            '0' after 80ns,
            '1' after 140ns,
            '0' after 180ns;

                  CONTINUED⇒
```

```
      X1 <= '0' after 10ns,
               '1' after 80ns,
               '0' after 140ns;

      X2 <= '0' after 10ns,
               '1' after 80ns,
               '0' after 100ns,
               '1' after 140ns,
               '0' after 180ns;

      X3 <= '0' after 10ns,
               '1' after 140ns,
               '0' after 220ns;

      RESET <= '0', '1' after 300ns;

      CLR <= '0', '1' after 5ns,
               '0' after 10ns,
               '1' after 200ns,
   process(CLK)'0' after 205ns;
   begin
      if (CLK = '0') then
         CLK <= '1' after 10ns;
      else
         CLK <= '0' after 10ns;
      end if;
   end process;

   process(RESET)
   begin
      if (RESET = '1') then
         terminate;
      end if;
   end process;
   end preg_test;
```

Figure 1: VCOMP Test Bench Output for Shift Register

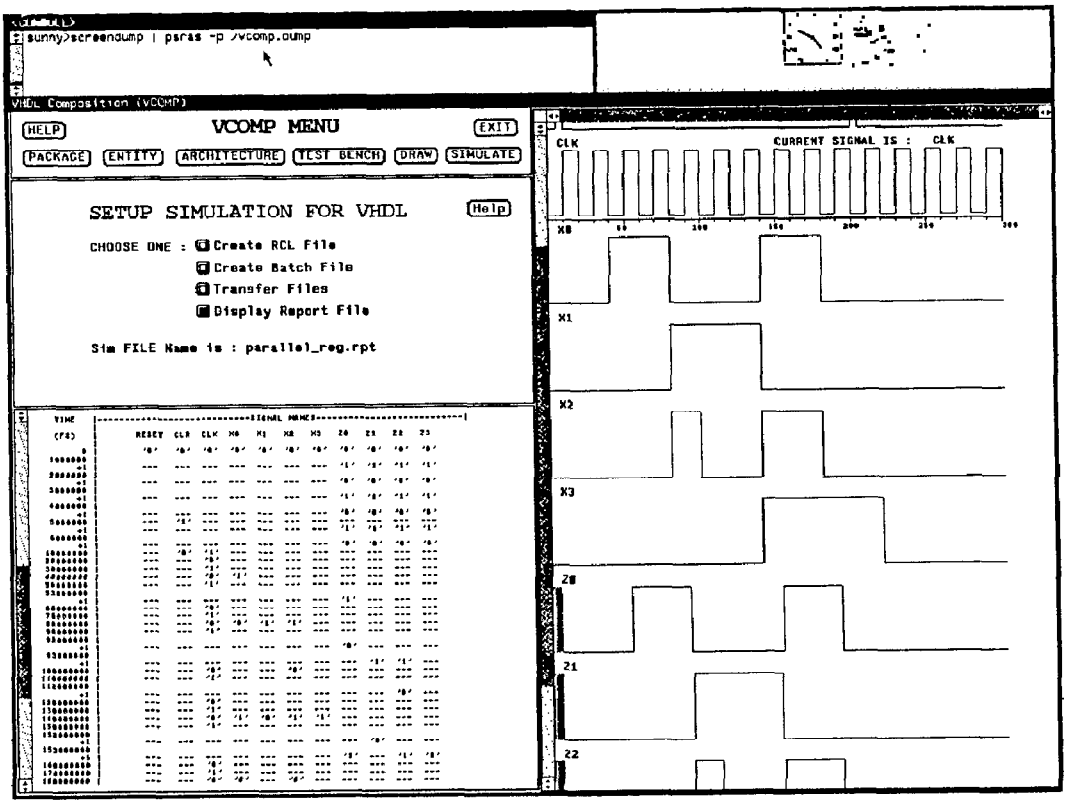

Figure 2: VCOMP Screen Display

waveform generated by VCOMP is shown in the screen display of Figure 2. Through the use of the *simulation* section of VCOMP, the report from the Intermetrics simulation of this register is easier to read and verify. While this example represents a relatively simple use of VCOMP, it illustrates several features of the system. Primarily, a designer can use the system to interconnect modules, prepare and run a simulation, and finally view the results for quick analysis and verification.

6. Conclusions

This paper has presented a system to improve the development of designs with VHDL. This system, the VHDL Composition Tool, uses VHDL and a support environment for VHDL as the basis for the software development system. Through the use of such a system, a hardware designer can quickly create a description of a system, formulate test vectors for the stimulus to the system, run a simulation of the system, and view the output reports to verify correct operation. Also, the designer, while not needing a complete understanding of VHDL and its support environment, can learn the syntax and specifications of VHDL as the interactive, graphics–based system is used.

While the system may be limited in the syntax of VHDL provided, VCOMP covers many of the necessary steps which are needed by a hardware designer. A more experienced VHDL programmer may need to use only a portion of the system for creating and viewing a design, but it is these features of the system which can be an aid to any hardware designer who needs quick verification of a design or system.

Acknowledgements

Funding for the development of this system was provided by the Virginia Center for Innovative Technology. Support for related research was provided by Research Triangle Institute.

REFERENCES

1. Waxman, R., "Hardware Description Languages for Computer Design and Test," *Computer*, Vol. 19, No. 4, April 1986, pp. 90–97.

2. Roth, G., "Standards, plus tools to adopt them equals integration," *VLSI Systems Design*, Vol. 8, No. 11, October 1987, p. 18.

3. *IEEE Standard VHDL Language Reference Manual*, The IEEE, Inc., New York, 1987.

4. Lipsett, R., Marschner, E., and Shahdad, M., "VHDL – The Language," *IEEE Design & Test*, Vol. 3, No. 2, April 1986.

5. Leonard, M., "VHDL–based simulator boosts design productivity tenfold," *Electronic Design*, Vol. 18, No. 7, May 12, 1988.

6. Sullivan, R. and Asher, L., "VHDL for ASIC Design and Verification," *VLSI Systems Design 1988 Semicustom Design Guide*, 1988, pp. 64–72.

7. *User's Manual for the Standard VHDL 1076 Support Environment (Draft)*, Intermetrics, Inc., Bethesda, MD, May 1988.

8. Marschner, E., "A VHDL Design Environment," *VLSI System Design*, Vol. 9, No. 9, September 1988, pp. 40–49.

9. Klenke, R., *A Graphic Representation of VHDL Models*, Master's Thesis, Department of Electrical Engineering, University of Virginia, Charlottesville, VA., January 1989.