A Visualization Framework for VHDL Analysis

C Costi and D.M. Miller
Department of Computer Science
University of Victoria, B.C. Canada
ccosti,mmiller@csc.uvic.ca

Abstract

We present a framework for visualizing structural and functional behavior in a HDL description. The intent is to assist designers in module reuse. Our approach is based on the perusal of the HDL code and employs signal and function analysis upon which new views of the module are based. A prototype software tool, VALET, which we are developing is described. While our tool is specifically targeted to VHDL descriptions, the overall approach can be adapted to alternative HDLs.

1. Introduction

As described in [1], there are three commonly used models for managing reuse of hardware intellectual property (IP) blocks. In the *IP-based reuse* model, a dedicated design team implements a component using specific design for reuse rules. This component is shared by different design teams. Sometimes, a technical department is in charge of supporting new users of a component. In the *rework-based reuse* approach, a block is designed for a specific project and after completion a dedicated team re-engineers the design to make it reusable and the component is included in the company's IP repository to make it available to other design teams. Similarly, in the *as-is reuse* model, a component is designed as part of a specific development project and put in the company IP repository without modification. The quality and reusability of the component depends on the design effort of the specific design team involved in the original project.

Using the first or second approachs, a design center may achieve a high level of reusability of its IP, but investments in resources, knowledge, time and reorganization are required. For these reasons, the *as-is reuse* approach is often followed and, unless specific discipline is enforced on using design for reuse rules, the company IP repository ends up including legacy designs. Legacy designs are components implemented with no design for reuse in mind. They are often poorly documented, but have valuable IP content. Teams wanting to reuse designs need to identify the original design intent directly from the component description.

Reusing an IP component means to select an appropriate block from among the available components in the repository. The objectives are to identify which component best matches the requirements of a new project, and to integrate it in the new design. Selecting a component from a repository is a difficult task and many research works exist on repository organization and automatic retrieval [2,3,4]. Integrating a component in a new design requires a detailed knowledge of the component interface. Moreover, changes must often be made so that the selected component will meet the specifications of a new design. In this case, knowledge of the component's structure and function is required.

Our research, progress on which is reported in this paper, aims to aid designers in the tasks of identifying the original design intent of legacy components and of acquiring enough knowledge of a reused block to allow for modification. We use a novel approach of analyzing VHSIC Hardware Description Language (VHDL) descriptions which allows a designer to investigate the function associated with a description. Our emphasis is on understanding the behavior of the hardware associated with a synthesizable HDL description and not the HDL code itself. The research reported here builts on our earlier work[5].

This paper is organized as follows. In the next section we review earlier research on the use and analysis of VHDL for reuse. Section 3 describes our methodology and the extraction of abstract concepts. Section 4 and 5 describe two viewer interfaces available in our prototype tool, and in session 6 the paper concludes with suggestions for further research and development of the VALET tool.

2. Related work

A method for the reuse of VHDL components is described in [6]. It is based on an object-oriented extension of VHDL: Objective VHDL. A reuse management system is presented which handles classification, modification, storage and retrieval of reusable components. No explicit analysis of the VHDL description is performed but specific features of Objective VHDL are exploited for design reuse.

Other researchers [7,8] have analyzed VHDL code using a technique called *slicing* which was first introduced by Weiser [9] in the software-engineering domain. For a signal

assignment line in a VHDL description, it is possible to extract a slice which consists of all the code lines which directly or indirectly affect the assignment execution and value. The slice represents executable VHDL code which behaves like the original code with respect to the chosen signal assignment. The authors showed that slicing may be used to extract a portion of code to be reused and to identify code which is influenced by a modification. While slicing explicitly identifies relation among lines of code, no interpretation of the meaning of the code is obtained.

A more semantic driven approach to VHDL code analysis is presented in [10]. The goal is to identify properties of the final circuit from the synthesis and testability points of view. The analyses presented identify signals which are implemented as memory elements in the final circuit, and signals that can be used to propagate test patterns through a portion of the design. These analyses cannot be directly used to understand the function of the component.

Understanding VHDL code is the goal of the research work reported in [11] where a VHDL reverse-engineering tool-set called VYPER! is presented. In VYPER!, a set of interfaces graphically represent information about the VHDL code including: (i) the control flow of concurrent statements; (ii) the control structure of sequential descriptions, and (iii) the module hierarchy. By means of hyperlinks, the designer may readily investigate the VHDL code and search for information which is necessary to reuse or modify the components under analysis. The goal is to help designers in analyzing and understanding the code itself and there is no attempt to identify or assist designers in understanding what function the VHDL code describes.

3. Proposed methodology

The goal of our approach is to assist designers in understanding the functional behavior embedded in a synthesizable VHDL description. Therefore, we are not focused on helping designers in understanding the structure of a VHDL description, but rather concentrate on techniques which extract functional information. We have identified classes of functional information as a set of abstract concepts which may be directly extracted from VHDL descriptions. An *abstract concept* represents a specific behavioral pattern which may be commonly found in a digital design. Considering static information, like control and data dependencies, each process in an architecture description, is decomposed into a set of primary abstract concepts, called *signal concepts.* These signal concepts are collected in groups to form larger concepts. Reviewing those concepts assists designers in identifying functional behavior in the VHDL description.

Manually inspecting the VHDL code to retrieve concepts is a very difficult and time consuming activity, especially when the design consists of hundreds or thousands of lines of code. Moreover, concurrent statements in VHDL introduce a high degree of complexity to the task since relations amongst signals may be spread across many different processes. Therefore, we are developing a prototype software tool, VHDL Assistant Low Efforts Tool (VALET), which assists designers in analyzing synthesizable VHDL descriptions.

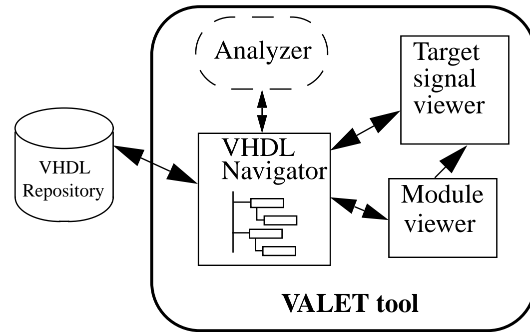

Figure 1. VALET structure

The structure of VALET is shown in Figure 1. The VHDL Navigator serves as the central controller in the tool. Once a component from the repository is loaded, it shows all VHDL structural elements in the design including instances, architectures, processes, ports and internal signals. From the VHDL Navigator, a designer may invoke incremental analyses which extract different abstract concepts. Currently, the automatically extracted abstract concepts may be examined using the *target signal viewer* and the *module viewer*. The signal viewer shows all concepts which refer to a specific output or internal signal which has been the target of at least one assignment. The module viewer shows concepts which interact to form a specific combinational or sequential hardware behavior.

3.1 Signal concepts

An algorithm called *signal analysis* is executed on each process in an architectural description to automatically extract a set of signal concepts. A *signal concept* consists of a target signal, a controller, a condition and a set *V* of values. The *controller* is a predicate whose values define when and which value in the set *V* is assigned to the target signal. The controller may be an input or an internal signal, or an expression like a relational expression or a function call. The condition consists of a Boolean expression which must be satisfied to enable the controller. The condition may be always true, which means that the controller is always active. Therefore the signal concept is always valid. Moreover, the condition may include the edge of a clock signal. Signal analysis recognizes a clock signal by using the pattern rules described in the IEEE P1076.6 standards document [12].

By examining a number of synthesizable VHDL descriptions for different applications, we defined a set of signal concept types which correspond to commonly found signal usages. Each type is named after the activity of the controller signal:

- *Enabler*. For a specific value or a range of values of the controller, the target signal is assigned to a value. For all other values of the controller there is no assignment to the target signal.
- *Set/reset*. For a specific value or a range of values of the controller, the target signal is assigned to a constant value *val*. For all other values of the controller, and at least one, a value which is not a constant is assigned to the target signal.
- *Tristate enabler*. For a specific value or a range of values of the controller, the target signal is assigned to values different from high impedance. For all other values of the controller, the target signal is assigned to high impedance
- *Tristate disabler*. For a specific value or a range of values of the controller, the target signal is assigned to high impedance. For all other values of the controller, values different from high impedance are assigned to the target signal.
- *Input choicer*. For some but not all values of the controller, the target signal is assigned to a value different from high impedance. For all other values of the controller, there is no assignment to the target signal.
- *Single assignment*. For all possible values of the controller, the target signal is assigned to the same value.
- *Selector*: For all possible values of the controller, the target signal is assigned to some value.
- *Partial selector*. For some but not all values of the controller, the target signal is assigned to some value.

Another special signal concept has been defined, called *logic net*. It is composed of a target signal, a value and a condition. It represents an assignment to a target signal which does not depend on any predicate. The condition is generally always true or represents a clock edge.

Considering a process, signal analysis extracts a set of signal concepts by trying to assign one of the above signal concept types to each target signal, predicate and condition. Since some of the above type definitions overlap, the classification is made in the order indicated above. That is, as soon as a type match is identified, no further search is performed. Note that given a target signal and a predicate, more than one signal concept may exist in the extracted set with that target and predicate. However, each of those signal concepts differ from each other by the controller expression.

3.2 Larger concepts

Another analysis algorithm, called *functional analysis*, examines the previously extracted signal concepts and identifies, when possible, larger abstract concepts. By looking at conditions of signal concepts belonging to different processes in the same architecture, functional analysis merges some of those concepts to form one of the following concepts:

- *Encoder*: consists of an *output o*, a *input i* and an *activation condition*, *i.e.* a Boolean expression. For each possible value of *i* a constant value is assigned *o*. The number of bits in *i* is greater or equal to the number of bits in *o*.
- *Decoder*: consists of an *output o*, a *input i* and an *activation condition*, *i.e.* a Boolean expression. For each possible value of *i* a constant value is assigned *o*. The number of bits in *i* is less than the number of bits in *o*.
- *Multiplexer*: consists of an *output o*, a *selector s* and an *activation condition*, *i.e.* a Boolean expression. For at least *k* possible values of *s*, *o* is assigned to a different $i \in I$ where the set *I* is composed of all port input and internal signals in the architecture. The value *k* is defined by a designer using a parameter *md* which we call *multiplexer degree,* $k = md \times values(s)$, where *value(s)* is the number of possible values of s. For *md* equal to 1, the functional analysis tries to find a complete multiplexer behaviour where for each value of *s* a different signal is assigned to *o*. A designer may freely choose *md*, between 0 and 1, provided *k* is greater than or equal to 2.
- *Demultiplexer*: consists of an *input i,* a *selector s* and an *activation condition*, *i.e.* a Boolean expression. For at least *k* possible values of *s*, *i* is assigned to a different target signal $o \in O$ where the set *O* is composed of all port output and internal signals in the architecture. The value *k* is defined by a designer using a parameter *dd* which we call *demultiplexer degree,* $k = dd \times values(s)$, where *value(s)* is the number of possible value assignments of *s*. For *dd* equal to 1, the functional analysis tries to find a complete demultiplexer behaviour where for each value of *s*, *i* is assigned to a different signal *o*. A designer may freely choose *dd*, between 0 and 1, provided *k* is greater than or equal to 2.

Intuitively, the above concepts represent combinational behaviors that are commonly found in digital design circuits. It is important to note that concepts recognized by the functional analysis do not necessarily constitute only combinational components which are actually synthesized. In fact the identified abstract concept may represent a behavior which is made up of separately synthesized elements. For example the VHDL code in Figure 2 is

```
ENTITY test_synop IS
  PORT ( A : in std_logic_vector(1 downto 0);
           Z : out std_logic_vector(1 downto 0));
END test_synop;

ARCHITECTURE mixed OF test_synop IS
SIGNAL value : std_logic_vector (7 downto 0);
BEGIN
  proc1: PROCESS (A)
    BEGIN
      IF (A = "01") THEN Z <= "11";
      ELSIF (A = "10") THEN Z <= "00";
      END IF;
  END PROCESS;
  proc2: PROCESS (A)
    BEGIN
      IF (A = "11") THEN Z <= "11";
      ELSIF (A = "00") THEN Z <= "10";
      END IF;
  END PROCESS;
END mixed;
```

Figure 2. VHDL encoder example

synthesized by design compiler synthesis tool (Synopsys) as combinational logic with two latches, while our functional analysis identifies an encoder concept with target signal *Z* and controller *A*.

4. Target signal viewer

The target signal viewer is an interface which allows designers to analyze all identified abstract concepts for a specific target signal.

A list of all target signals used in an architecture is presented to the user. By selecting a target all recognized abstract concepts associated with the specific target are presented as a list (*L*). At this point, the user may investigate the list *L* of abstract concepts using different methods.

4.1 Analysis of a specific abstract concept

The user can select a specific abstract concept in the list *L* and view the type and the condition which activate the concept. Moreover, the user can choose a specific value for each controller and view the behavior of the concept, that is which value is assigned to the target signal.

4.2 Filter and group by controller

The user can choose a controller *C* which is used by at least one of the concepts in *L* and create a group *G* containing only those concepts which have the controller *C*. Once the group *G* is formed, the user can invoke the *extra controls recognition* analysis which identifies new predicates which control the activation of the abstract concepts in the group. This analysis looks at conditions associated with each abstract concept in *G* and at terms in conditions which refer to one of the input or internal signals in the group. Then, it identifies and classifies those signals which satisfy one of the following condition:

- *Extra selector*. For each possible value of the signal, an abstract concept in the group *G* is activated. Moreover, all concepts in the group must be activated by at least one value of the signal.
- *Extra partial selector*. For more than one but not all possible values of the signal, an abstract concept in the group *G* is activated. Moreover, all concepts in the group must be activated by at least one value of the signal.
- *Extra enabler*. For a specific value or range of values of the signal, all concepts in the group *G* are activated.

As the result of the extra control recognition analysis, the group *G* is enriched by a set of extra control signals of types *extra selector*, *extra partial selector* or *extra enabler*. At this point, users may interact with the group by choosing values for the controller *C* and each extra control signal and view its behavior, that is which concept is activated and which value is assigned to the target signal.

4.3 Equivalences

Except for a degenerate case, the list *L* of abstract concepts contains concepts which refer to the same target signal but have different controllers. It can happen that different controllers influence the same set of assignments. That is, a set of assignments to the target signal representing a particular behavior may be seen as controlled by values of either one controller or another controller. For example, two different selector signal concept may be identified for the fragment of VHDL code in Figure 3:

- Selector concept with target signal *Z*, controller *A*, condition *B='0'* and values *C AND D* for *A='1'* and *E* for *A='0';*
- Selector concept with target signal *Z*, controller *B* and condition *A = '1'* and values *C AND D* for *B='0'* and *E* for *B='1'*.

Obviously the two selector concepts represent the same behavior, it is only the point of view, and in particular the controller, that is changed.

An algorithm called *equivalence analysis* is part of the target signal viewer. Once a user selects one or more abstract concepts in *L*, the algorithm seeks to identify all possible groups of concepts which form equivalent groups within the group of selected concepts.

```
IF (A = '1) OR (B = '0') THEN
    Z <= C AND D;
ELSE
    Z <= E;
END IF;
```

Figure 3. Equivalence

4.4 Custom group

The target signal viewer allows the user to manually create and save a group which consists of a subset of abstract concepts in L. After having examined the list L of abstract concepts using one of the above methods, a user may decide to create a personal group. In fact, the user may decide to form a single group of concepts which have different controllers but whose behaviors are in fact closely associated and together provide a better understanding of the module's behavior. While adding concepts to such a group the viewer interface advises a user when all possible assignments for the target signal have been covered by the concepts in the group.

5. Module viewer

The module viewer is an interface which shows the results of the *partition analysis* algorithm for a specific architecture. This algorithm considers the set S of all extracted abstract concepts for a selected architecture and, by looking at data dependencies among them, partitions the set S into a set P of non overlapping subsets P_i of S. Each P_i corresponds to one of the following module definition:

- *Fsm module*: is identified by abstract concepts with *state* as target signal. State signal requires a memory element to maintain its value, and its value is controlled by its own previous value. It means that there is at least one value of the state signal which, possibly together with other signal values, determines a new value for itself.The module also consists of a list of target signals controlled by the state signal.
- *Complex register module*: is identified by one or more *complex register* signals which require memory elements to maintain their values. Moreover, there is at least one value of one signal, which is different from any complex register, which causes the value of a complex register to depend on its previous value. But unlike the fsm module, there is no value of any complex register which determines a new value for itself. Abstract concepts with *complex register* signals as target are collected in the module.
- *Simple register module*: consists of one or more *simple register* signals which require memory elements to maintain their values. Their values do not depend in any way on their own previous values. Abstract concepts with *simple register* signals as target are collected in the module.
- *Combinational module*: consists of abstract concepts with a target signal which does not require a memory element.

Each module represents a different degree of complexity of behavior. The module names have been chosen to resemble a common hardware behavior associated with the behavior identified. The user may select one of the P_i modules and explore its content. Depending on the type of module a different interface is provided which explicitly indicates and describes the role of each of the module's input, output and internal signals. Each signal in the module which is also a target in the architecture may be examined by selecting it and open a target signal viewer for it.

6. Future work

The research reported upon here is ongoing. More work is needed on the module viewer interface to show more details for each module. We plan to provide a graph with data dependencies and allow the user to navigate that graph and to interrogate the module. Currently analyses are only performed inside an architecture. We are interested in examining interaction amongst multiple architectures.

7. References

[1] M.Keating, P.Bricaud, "Reuse Methodology Manual for System-On-A-Chip", Kluwer Academic Publisher, 1998.

[2] J.Altmeyer, S.Ohnsorge, B.Schurmann, "Reuse of Design Objects in CAD Frameworks" in Proceedings International Conference Computer Aided Design, 1994.

[3] M.Koegst, P.Conradi, D.Garte, M.Wahl, "A Systematic Analysis of Reuse Strategies for Design of Electronic Circuits" in Proceedings Design Automation and Test in Europe, France 1998.

[4] A.Reutter, W.Rosenstiel, "An Efficient Reuse System for Digital Circuit Design" in Proceedings Design Automation and Test in Europe, Germany 1999.

[5] C.Costi,M.Miller, "A VHDL analysis environment for design reuse" in Proceedings second International Forum on Design Languages (FDL), Lyon, France 1999.

[6] C.Barnas,W.Rosenstiell, "Object-oriented reuse methodology for VHDL" in Proceedings Design Automation and Test in Europe, Germany 1999.

[7] M.Iwaihara, M.Nomura, S.Ichinose, H.Yasuura, "Program Slicing on VHDL Descriptions and Its Applications" in Proceedings 3rd Asian Pacific Conference Hardware Description Languages, Bangalore 1996.

[8] E.M.Clarke, M.Fujita, S.P.Rajan, T.Reps, S.Shankar, T. Teitelbaum, "Program Slicing for Design Automation: An Automatic Technique for Speeding-up Hardware Design, Simulation and Verification" Technical Report, Computer Science Department University of Wilsconsin 1998.

[9] M.Weiser, "Program slicing" in IEEE Transactions on Software Engineering, 1984.

[10] L.Baresi, C. Bolchini, D. Sciuto "Software Methodologies for VHDL Code Static Analysis based on Flow Graphs" in Proceedings Design Automation and Test in Europe, 1996

[11] Gunther Lehmann and Bernhard Wunder and Klaus D. Muller-Glaser, "Basic Concepts for a HDL Reverse Engineering Tool-Set" in Proceedings International Conference Computer Aided Design, California 1996.

[12] IEEE, *"IEEE P1076.6 Standard for VHDL Register Transfer Level Synthesis,"* New York, NY, I1999.