VHDL Structural Model Visualization

Dominik Macko, Katarína Jelemenská
Slovak University of Technology
Faculty of Informatics and Information Technologies
Bratislava, Slovakia
dikemdm@gmail.com; jelemenska@fiit.stuba.sk

Abstract— **Nowadays the digital systems design is almost exclusively realized using hardware description languages (HDL). In Europe, the VHDL (Very-High-Speed Integrated Circuits HDL) is the most widely used HDL. Although HDLs brought a lot of advantages into the design process, the HDL structural models, especially on register transfer and lower layers, are harder to read than the previously used schematic representations. That is why a lot of commercial EDA (Electronic Design Automation) systems include some kind of visualization tool enabling to represent a model structure in a graphical form. However, these systems are usually too complex and expensive for the education purpose. In this case simple, easy to use visualization tool would be more appropriate. The paper deals with the problem of the structural models visualization as well as with the design and implementation of the visualization tool devoted to the VHDL structural models visualization. The presented tool offers the possibility to display the schematic view corresponding to the input VHDL model, edit the schematic layout, print it or export it to an image file. The tool preserves the model hierarchy and enables to easily switch among the respective levels. It represents an useful tool in the process of the VHDL structural model verification and debugging as well as for documentation purposes.**

Keywords- VHDL; structural models; visualization;

I. INTRODUCTION

The VHDL (VHSIC Hardware Description Language) standardization in 1987 has started the massive use of the language itself as well as the massive development of EDA (Electronic Design Automation) tools supporting it. The digital system design based on HDL brought several advantages compared to the previously used manual design style – clearer designs with less mistakes, possibility to verify the design in early stages, technology-independence, and high productivity increase thanks to the available EDA tools. On the other hand, HDL design is less illustrative for a human being, especially in case it represents the digital system structure. In general it is easier to follow the circuit structure in a graphical form than in a textual one, therefore easier to detect the possible mistakes in a design (e.g. caused by an incorrect interconnection). The VHDL model visualization is useful not only for verification purposes, but also for design documentation. Adding the graphical representation of the designed model allows other designers to understand the design structure easier and faster. Finally, it can also be useful when teaching the VHDL modeling techniques, where the model visualization can help the students to imagine, how the VHDL model works.

II. RELATED WORK

Nowadays, a lot of professional automatic design tools provide several ways to describe digital system behavior and structure together with its optimization. They support various types of graphical and textual editors e.g. to enter the state diagrams, truth tables, logic diagrams or simply to describe the circuit in HDL. However, most of the EDA tools supporting structure visualization are the commercial ones. We can mention for example HDL Author, Visual Elite, or Active-HDL [1-3].

HDL Author is the product of Mentor Graphics [1] that replaced the previously supported HDL Detective. The HDL code visualization is just one of its many functions and enables HDL code conversion into a Block Diagram, IBD, State Diagram or Flow Chart. The objects layout, color, style, and font can be modified in a way that will not influence the original code in any way. However, HDL Author is a robust application, which is very complex and quite expensive.

Visual Elite was originally developed by Summit Design and later bought by Mentor Graphics [2]. This is another complex and general application that offers a variety of HDLs and other design entry methods including block diagrams, state diagrams and connectivity tables. The automatic mapping of source code into its graphical representation is also provided. The quality of code visualization is not that high compared to the HDL Author, however, the user interface is easier to manage by new users.

Another complex design system supporting HDL code visualization is Aldec's Active-HDL [3]. It is a suite of many tools covering all phases of FPGA design and verification. The Code2Graphics tool enables the HDL code conversion into its graphical representation that can be further edited. Unlike in HDL Author or Visual Elite, the modifications can alter the model itself and the graphical representation can then be converted back to the HDL using the Graphics2Code tool. Moreover, Active-HDL also supports the visualization of simulation. During the simulation an actual state of every module in the block diagram is visualized and the values of all variables are visible in the waveforms. In many other ways Active-HDL is comparable to Visual Elite.

All the mentioned tools and systems are the commercial ones. They all provide many functions (including visualization) and support several HDLs. The main issue is their excessive complexity and the price which is often not affordable for

This is an accepted version of the published paper:
D. Macko and K. Jelemenská, "VHDL structural model visualization," 2011 IEEE EUROCON - International Conference on Computer as a Tool, Lisbon, 2011, pp. 686-689.
doi: 10.1109/EUROCON.2011.5929348
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5929348&isnumber=5929030

beginners and students – the type of users that need the structure visualization the most. This was the reason why we decided to dedicate our effort to the development of simple and easy to use tools for HDL structural description visualization. Several solutions have been developed at the Faculty of Informatics and Information Technologies, Slovak University of Technology (FIIT STU) in Bratislava, most of them in the frame of diploma theses [4 - 6]. The work presented in this paper belongs to this number.

III. VHDL Model Visualization Requirements

To visualize the VHDL model it has to be syntactically analyzed first. The context-free parser is then necessary to transform the VHDL description into a transient format. The designed transient format should be able to represent also the visualization information (e.g. location or size). We have to consider all the elements that should be displayed in the graphical representation. The entity instance will be represented by a shape, holding the instance name and the ports that will be connected to the ports of other instances. The signals will be displayed as the connection lines, joining the corresponding ports. The signals visualization should be designed in such a way that the minimum number of intersections is reached. It is also useful to preserve the model hierarchy in the graphical representation.

The visualization tool should offer the following features:

- Loading the VHDL model from a file and its transformation to the schematic representation.
- Separated view of individual levels of hierarchy.
- Objects layout modifications at the selected level of hierarchy – the original interconnections have to be preserved.
- Export of the currently displayed hierarchical level into an image format.
- Saving the modified layout into a file for further usage.
- Simple user interface –intuitive, easy to use etc.

IV. Visualization Tool Design

To analyze the VHDL model the parser generator ANTLR v3 [7 - 9] is one of the possible solutions. Based on the input grammar the parser generator generates a group of C# classes (another programming language can also be chosen) performing the VHDL code analysis. The VHDL grammar description preparation as well as the parser classes generation are one-time steps which will be repeated only in case the grammar has been modified.

For further use of the analyzed VHDL description it is suitable to create an object model that will keep the hierarchical structure of the VHDL model. The information necessary for the model visualization (e.g. location or size of the individual objects) will also be included in the object model. The model contains only the data directly necessary for the visualization purpose. It does not include information present in VHDL model that is not necessary for this purpose.

A. Transient Representation of VHDL Model

The model, created in this way can be saved to a transient representation, suitable for visualization. An XML representation does have the appropriate properties for this purpose since it can preserve the hierarchical structure. It is widely used which means there is a lot of classes and libraries available to work with the XML file format. The similar solutions were used in [10]. In terms of visualization, there are 3 types of objects defined in the XML file: Entity instance, Port, and Connection.

1) Entity instance

The entity instance is the only object in visualization that can have its own structure. In case it has the internal structure defined, it is described by means of other objects – other entity instances, which are described in the lower level of hierarchy. For each entity instance, the following parameters are extracted from the VHDL description: Entity instance name, Entity declaration name, Architecture name, and Ports list.

2) Port

Every entity instance keeps its ports list. Unlike in the VHDL description, the ports just illustrate the interfaces, by means of which the instance in the real design communicates with the other ports. Each port has the following parameters: Port name, Port mode, and Port type.

3) Connection

The connection joins two ports and presents their ability to communicate to each other by means of sending signals. Each entity instance keeps its own connections list. If it is the entity instance on hierarchical level n, its connections list describes the connections between the ports of the entity instances on the level n-1, describing thus the structure of the given entity instance on the lower level. The connections list includes also the interconnections among the given entity instance ports (on level n) and its lower level structure (ports on level n-1). Each connection has the following main parameters: Connection name, Source port name and entity instance name, Destination port name and entity instance name, Source port range name, and Destination port range name.

To preserve the hierarchical structure of the original VHDL description it is sufficient to keep the information about the architectures in the VHDL description, along with their ports, connections and the entity instances inside. Based on this information we can create the hierarchical structure of the VHDL model for visualization purpose. In the design of XML schema 4 types of nodes have been defined: Architecture, Port, Entity instance, and Connection. Using this transient XML representation it is possible to visualize the structural VHDL model.

B. Visualization of XML Representation

Since the XML representation preserves the hierarchical structure of the VHDL model, it is not difficult to visualize the schematics of the respective levels of hierarchy. The architecture in the XML file represents one hierarchical level. In fact, it is an entity instance at the upper hierarchical level. Its description includes the ports of the given entity instance (upper level ports), the entity instances of the given level, their

ports, and the interconnections among the ports. So it is necessary to design the graphical representation of each object of this architecture.

The port can have different modes (in, out, inout, buffer, linkage). According to the port mode, the shape of port is drawn (see Fig. 1). The direction of the port shape (arrow) symbolizes the direction of the data flow. The ports can belong directly to an entity instance. In that case, they allow the entity instance to communicate with other instances. If the port does not belong to any instance, it represents input to or output from the upper level of hierarchy. It is also necessary to consider the special ports, which can represent the constant value sources or the globally defined signals.

The entity instance is visualized as so called "black box", which functionality we do not know. So it is represented by a rectangle (see Fig. 2), displaying the entity input and output ports, the entity name, and the name of the respective instance.

The connection is represented as a broken line, which connects two ports (see Fig. 3). If the connection interconnects the ports of the internal instances, the connection contain also the signal name (Fig. 3a). However, the connections can pass from one port to several ports or vice-versa. So it is necessary to define, which connections can overlap or how the connections relations should be represented. The schematic representation of the VHDL model should satisfy the standard notation. Therefore the branching and splitting of the connection has to be defined as well. The connection branching is the case, when the same signal passes to several ports (Fig. 3b). The connection splitting/joining is the case, when the multi-bit signal splits into several one-bit signals (Fig. 3c), or several one-bit signals join into one multi-bit signal (Fig. 3d). Next to the multi-bit ports there is the number of bits displayed.

C. *Objects Layout Algorithm*

The algorithm used for model visualization locates the entities in two rows. In the upper row, the odd entities and in the lower row the even ones are displayed. The entities are numbered according to their sequence in the VHDL model. The width of the entities is constant, and the height is automatically calculated based on the number of ports. The row distance from the top of the display surface is also constant.

So is the distance from the left border. The entities are drawn

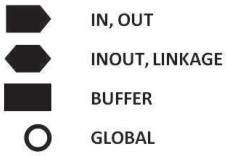

Figure 1. The port graphical representation dependent on the port mode.

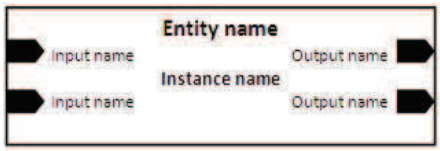

Figure 2. The graphical representation of an entity instance.

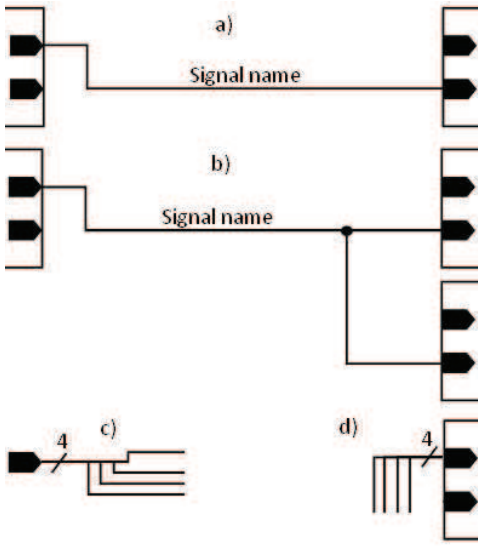

Figure 3. The representation of a) connection, b) branching, c) splitting, and d) joining.

gradually up down and left to right. Even the horizontal distance between the entities is constant. The vertical distance between the entities is calculated according to the number of connections and the upper row height, which is given by the highest entity in the row.

In the displayed hierarchical level, the ports are divided into external and internal. The internal ports are those, which are situated inside the entity instances of the given hierarchical level. The external ports belong to the upper level entity. The ports have the constant width and height. The internal ports are automatically aligned to the borders of the given entity. The input ports are on the left hand side, the others on the right hand side. The vertical distance between the ports is constant too. The external ports are displayed near the left and right border of the display surface. The distance from the border is constant as well. The ports are aligned to the connections position.

There is a virtual bus located between the entities rows. The distances between the connections in the virtual bus are constant. Each inter-ports connection has its own part of virtual bus reserved. This ensures that the different connections will not overlap in the horizontal direction and it is no longer necessary to check this fact (algorithm speed-up). The vertical parts of the connections are checked, not to overlap one with another. If there is an overlap, the connection is shifted. The connections can overlap only in case they are connected to the same pin of the same port.

All the constant values used by this algorithm can be preset by user.

D. *User Interface of the Visualization Tool*

The user interface should be user-friendly. That means, is should be simple, intuitive, comfortable and offering the possibilities to access all the application features, to fulfill the requirements.

V. ILLUSTRATING EXAMPLE

To illustrate the features of the implemented visualization tool a simple example will be presented here. In the code given in Fig. 4, there is a fragment of simple VHDL model describing the structure of 3 bit shift register.

```
entity DFF is
  port (D,Clk:in bit;Q:out bit);
end DFF;

architecture Beh of DFF is
begin
  Q <= D when Clk'event and Clk = '1';
end Beh;
-------------------------------------------------
entity Shift3bitReg is
  port (In1,Clk:in bit;Out1:out bit);
end Shift3bitReg;

architecture Struct of Shift3bitReg is
  component DFF is
    port (D,Clk:in bit;Q:out bit);
  end component;
  signal s1,s2:bit;
begin
  dff1:DFF port map (s2,Clk,Out1);
  dff2:DFF port map (s1,Clk,s2);
  dff3:DFF port map (In1,Clk,s1);
end Struct;
```

Figure 4. VHDL model of three bit shift register.

This VHDL code, used as an input of visualization tool, is then analyzed and transformed using an object model into the XML representation. An example of XML representation of entity instance dff1 is given in the code fragment displayed in Fig. 5. The resulting visualization of the representation is shown in Fig. 6.

```
<EntityInstance x="670" y="50" width="250" height="60">
  <InstanceName>dff1</InstanceName>
  <EntityName PackageName="" LibraryName="">dff</EntityName>
  <ArchitectureName PackageName="" LibraryName="">beh</ArchitectureName>
  <VHDLPorts>
    <Port anchor="left" x="670" y="60" width="20" height="10">
      <PortName>d</PortName>
      <PortMode>IN</PortMode>
      <PortType>bit</PortType>
    </Port>
    <Port anchor="left" x="670" y="80" width="20" height="10">
      <PortName>clk</PortName>
      <PortMode>IN</PortMode>
      <PortType>bit</PortType>
    </Port>
    <Port anchor="right" x="900" y="60" width="20" height="10">
      <PortName>q</PortName>
      <PortMode>OUT</PortMode>
      <PortType>bit</PortType>
    </Port>
  </VHDLPorts>
</EntityInstance>
```

Figure 5. Fragment of VHDL model XML representation.

VI. CONCLUSIONS AND FURTHER WORK

The paper is devoted to the problem of visualization of structural digital system models described in VHDL. The core of the paper forms the design and implementation of the new VHDL visualization tool. The tool is especially suitable for the beginners in VHDL design and for educational purpose. It helps to understand the VHDL structural model development and is useful in the process of digital system design documentation. For a human being the graphical representation of the structural model, generated by this tool, is better to understand and easier and faster to detect the mistakes made during the VHDL structural model creation.

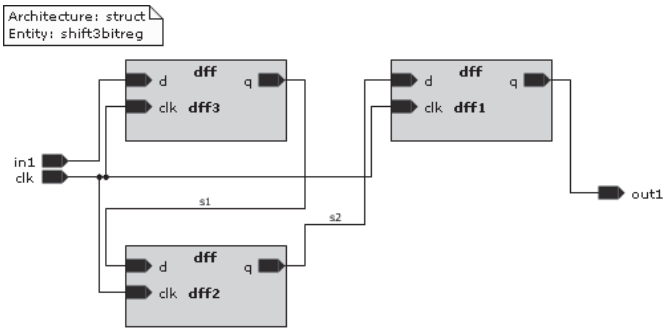

Figure 6. Visualization of a simple VHDL model.

The visualization tool is prepared for the extension by VHDL model simulation and visualization of this simulation. After this extension the tool would become also a strong verification tool, valuable in digital systems design.

ACKNOWLEDGMENT

This work was partially supported by Slovak Science Grant Agency (VEGA 1/0649/09 "Security and reliability in distributed computer systems and mobile computer networks").

REFERENCES

[1] Mentor Graphics, "Manage design data and flows - HDL Author," Mentor Graphics's products, Online, May 2009, 1-800-547-3000. http://www.mentor.com/products/fpga/hdl_design/hdl_author/

[2] Mentor Graphics, "Continuous design flow from TLM to RTL - Visual Elite HDL," Mentor Graphics's products, Online, May 2009, 1-800-547-3000. http://www.mentor.com/products/fpga/hdl_design/visual-elite-hdl/

[3] Aldec, Inc., "Active-HDL," Aldec's products, Online, May 2009. http://www.aldec.com/ActiveHDL/

[4] J. Petráš, "VHDL model visualization", master theses, FIIT STU Bratislava, 2008, 85 p.

[5] M. Zubal, "VHDL model visualization", master theses, FIIT STU Bratislava, 2008, 80 p.

[6] J. Turoň, K. Jelemenská, "Contribution to graphical representation of SystemC structural model simulation," in Proc. of the 7th FPGAword Conference, L. Lindh, V.J. Mooney, S. de Pablo, J. Öberg, Eds. Copenhagen (Denmark), September 2010, pp. 42–48.

[7] T. Parr, "Implementing parsers and state machines in Java" in Java VM Language Summit 2009, University of San Francisco.

[8] T. Parr, The Definitive ANTLR Reference: Building Domain-Specific Languages. The Pragmatic Bookshelf, 2007.

[9] R. M. Volkmann, "ANTLR 3," Online, October 2010. http://jnb.ociweb.com/jnb/jnbJun2008.html

[10] M. H. Reshadi, B. Goji-Ara, Z. Navabi,"HDML: compiled VHDL in XML" in VHDL International Users Forum Fall Workshop. Teheran Univ., 2000, pp. 69-74.