Visualization of Verilog Digital Systems Models

K. Jelemenská, M. Nosáľ, P. Čičák
Institute of Computer Systems and Networks, Faculty of Informatics and Information Technologies
Slovak University of Technology Bratislava
Ilkovičova 3
842 16 Bratislava, Slovakia
jelemenska@fiit.stuba.sk; nosal.michal@gmail.com; cicak@fiit.stuba.sk

***Abstract*— Nowadays the digital systems design is almost exclusively realized using hardware description languages (HDL). Verilog belongs to the HDLs that are the most widespread especially in the United States. However, the textual HDL representation of structural model is less understandable compared the schematic one. Therefore a transformation of the structural HDL description into its graphical schematic representation is a useful function for hardware designers. In this paper the HDL Visualizator is described that was designed and implemented to support this function for Verilog structural models. The paper addresses several problems of visualization process and their possible solutions. The design and implementation of visualization tool that is able to display the schematic view as well as the simulation results of structural Verilog model is also presented.**

I. INTRODUCTION

In the process of complex digital systems design hardware description languages (HDLs) have an irrecoverable role. They provide designers the possibility to describe the hardware behavior and structure on the various abstraction levels. However, the structural description in HDL form is often unreadable for the persons not familiar with this kind of design. In these situations the tools for graphical visualization of HDL structural description are very useful.

There are several applications, design tools or development environments that are capable of graphical visualization of a design structure. This possibility is usually only one of many other functions available in the expensive and often too complex solutions for certain group of users, e.g. students or beginners in HDL design. There are very few, if any at all, easy-to-use and affordable applications offering HDL models visualization especially when they are written in Verilog. Our aim was to overcome this gap.

The problem of visualization of digital system Verilog structural description is the main subject of this paper. The paper covers application design, its implementation and testing results.

II. RELATED WORK

The digital systems design methods have evolved dramatically over the last 30 years. The traditional design methods, based on mathematical equations or schematic approach to the hardware design, became impractical or even useless in the process of complex digital systems design.

Modern design methods based on HDLs and design automation tools brought the effectiveness into the design process and sped it up substantially. Another tremendous advantage was the replacement of prototyping by simulation. The designers can model hardware structure and behavior on various abstraction levels using HDLs and simulate the model to verify its properties. 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. The most commonly used HDL languages are VHDL, Verilog and SystemC.

There are many implementations tools with wide spectrum of functions to make the design process easier, more effective and faster. However, most of the software tools capable of structure visualization described in Verilog are the commercial ones. To the most widespread belong Visual Elite, Active-HDL and HDL Author [1-3].

HDL Author is the product of Mentor Graphics company [1] that replaced the previously supported HDL Detective. The main goal of this tool is the improvement of team work and data exchange between team members therefore it was designed for data management in all phases of design process. The HDL code visualization function is just one of its many functions and enables also Verilog code conversion into a Block Diagram, IBD, State Diagram or Flow Chart. The objects layout, color, style, and font can be modified i.e. the non-logical edits can be applied 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 company and later bought by Mentor Graphics [2]. This is another complex and universal application to design and build products. It offers a variety of HDLs and other design entry methods including block diagrams, state diagrams and connectivity tables. It also provides automatic mapping of source code into graphical representation. However, the quality of code visualization is not as high as in HDL Author. On the other hand the user interface is very comfortable and easier to manage by new users, than relatively complicated HDL Author's user interface.

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. In the context of this paper the tool called Code2Graphics is an interesting one. This tool enables the HDL code conversion into a graphic representation that can be further edited. Unlike in HDL Author or Visual Elite, the altered graphic representation can be converted back to the text form of

arbitrary supported HDL using 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 beginners and students.

Since the problem of lost clarity in textual description of structure is especially relevant to students and beginners, 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, 7, 8].

VHDLVizualizer [4] is the tool dedicated to the VHDL structural description visualization. The architecture of this solution consists of six components (see Fig. 1). The VHDL parser was built by ANTLR (ANother Tool for Language Recognition) that provides a framework for arbitrary language parser generation based on its grammar definition [5]. The graphic user interface is based on Netron Graph Library and an XML (Extensible Markup Language) format is used to represent the visualized VHDL description internally.

The tool is able to check the VHDL source code syntax and to visualize the VHDL structural description preserving its hierarchy. It also supports basic non-logical editing of the resulting visualization like modules layout modification, modules connections formatting, notes insertion etc. The created diagrams can be exported into jpeg files or saved in XML file. The main limitation of this solution was the quality of the displayed layout of modules and especially the interconnections which was far from ideal. This issue was discussed and partially solved in [6].

Another solution with the functionality similar to VHDLVizualizer was called *VHDL Visualizator* [7]. The system architecture is basically the same but this tool was implemented on Delphi and Java platforms. An internal representation of visualized structure is also based on XML format. Although the quality of visualization is a bit higher in this case, the tool supports smaller subset of VHDL.

An interesting solution represents [8] that was focused on the SystemC models visualization, based on the modifications of SystemC library itself. However, the SystemC nature is substantially different from other HDLs, therefore the approach adopted in this solution can not be used for Verilog models visualization.

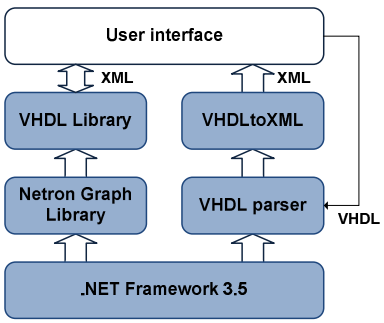

Fig. 1. The VHDLVisualizer architecture

The visualization tools already developed at FIIT STU Bratislava were devoted to VHDL and SystemC HDL languages. The third commonly used HDL – Verilog was not yet supported. This work is supposed to overcome this gap.

III. Visualisation of Structural HDL Description

In the context of HDL structural description visualization, the important language constructs are module definitions, port definitions, module instances and their mutual connections. In schematic representation each module is represented by a graphical entity with ports that are connected with ports of other modules. The visualization process is illustrated in Fig. 2. There are three main problems that have to be solved in the visualization process: syntactic analysis of Verilog code, internal data representation, and graphical layout of the visualized data.

The first step in the visualization process is a conversion of Verilog source code into its internal representation. During this step two issues have to be addressed: Verilog code *syntactic analysis* and *internal representation* selection. An appropriate internal representation should be able to keep the information about the modules and connections position, size and general layout. From this step the third issue results: the optimization of modules and connections *graphical layout*.

A. Syntactic Analysis of Verilog Code

Syntactic analysis is relatively complicated but an unavoidable step in the visualization process. However, there is a possibility to use an existing parser for this purpose. Several freeware tools are available supporting among other functions Verilog syntax checking [9]. *Icarus Verilog* is a tool for Verilog simulation and synthesis. *Source Navigator for Verilog* is an open source development environment for Verilog which has a built in parser. *CvSDL* represents C++ libraries for Verilog simulation. *GPL Cver* denotes another Verilog simulator with full support of IEEE 1364-1995 Verilog standard and partial support of Verilog 2001. *Verilog2C++* is a tool performing Verilog code transformation into its C++

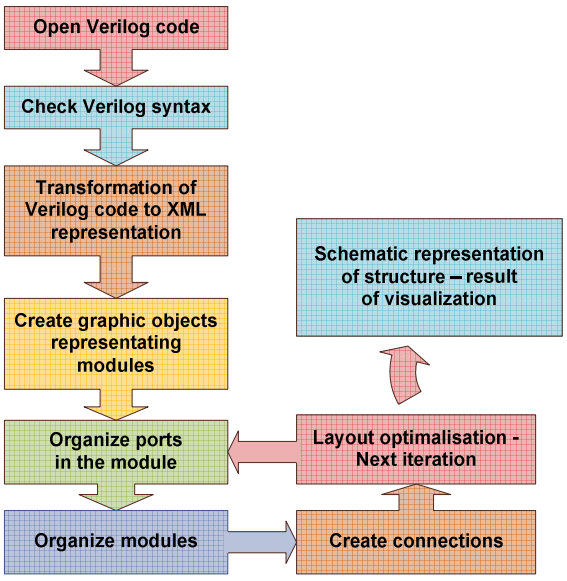

Fig. 2. The visualisation process of HDL structural description

equivalent. Finally, *Verilator* stands for Verilog simulator suitable for large projects, where simulation performance is the primary concern.

From the available freeware tools the *Icarus Verilog* was identified as the most suitable for our purpose, since in addition to the syntactic analysis it enables also to use the simulation results in our visualization tool.

The next step in the process of visualization is the transformation of Verilog description into its internal representation, what should be done by parser. Since we were not able to find a suitable open source parser for Verilog language, we were considering the available parser generators that can generate a specific language parser based on the language grammar definition. The parser generators like *Gardens Point Parser Generator* (GPPG) [10], *GRAMMATICA* – Parser Generator [11], *ANTLR v3* [5] and *Compiler Generator Coco/R* [12] have been analyzed and the *ANTLR v3* parser generator was chosen as the most suitable one for our purpose.

B. *Internal Representation of Structural Description*

There are two basic approaches for internal data representation: an object-oriented approach and the XML based approach. An example of the *object-oriented structure representation* format is AIRE/CE (The Advanced Intermediate Representation with Extensibility/Common Environment) [13], a specification originally designed for the representation of VHDL description that can potentially be used for Verilog description as well.

In general, the internal structure *representation based on XML* is more flexible than the object-oriented one. There are several specific XML schemes that could be used for our purpose including: *HDML* (Hardware Description Markup Language) [14] – an XML scheme created for hardware description, *VXML* [15] – the VHDL Markup Language scheme which is more suitable for VHDL description, *MoML* (Modeling Markup Language in XML) [16] – the XML scheme created for hardware design description like HDML. However, each of them would require some modifications to be tailored to Verilog language and our specific needs. Therefore we decided to design our own XML based representation. The reason was its higher flexibility and potential interoperability with third part XML editors. The resulting XML scheme is given in Fig. 3. The internal data representation will contain information about visualized structure as well as simulation results.

C. *Graph Drawing Algorithms*

Graph drawing, including optimization of modules and connections layout, is quite a complex problem. Several rules and algorithms are known for graph drawing and optimization that can be used for Verilog models visualization. The good graph look is very subjective issue, often it is just a matter of aesthetic taste, not the exact parameters. Therefore it is difficult to choose the best algorithm. In spite of this there are a few criteria for graph layout evaluation [17]:

Minimize crossing: keep the number of lines cross to a minimum.

Minimize area: keep the graph area to a minimum.

Minimize the sum of edge lengths.

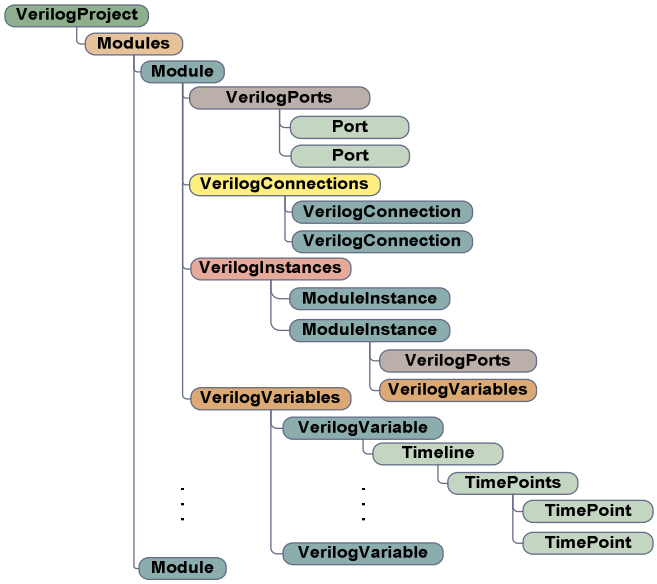

Fig. 3. The XML scheme of internal data representation

Minimize the difference between edge lengths: keep the edge lengths as same as possible.

Minimize bends: keep edge bends to a minimum.

IV. VISUALISATION OF HDL SIMULATION

HDL model simulation is much more complicated than HDL model structure visualization. A testbench has to be developed to simulate the Verilog digital system model.

In the visualization process of Verilog model simulation a few basic steps can be identified. The first one is the Verilog code integrity and syntax checking. The second step is simulation itself and the third one is the transformation of simulation results into the internal representation and their visualization. There is also the possibility to visualize the simulation results from external VCD[1] (Value Change Dump) file. In this case the integrity of this file must be checked as well.

V. SYSTEM ARCHITECTURE

Based on the previous analysis the system architecture was designed that is illustrated in Fig. 4.

The Icarus Verilog was used to perform the syntactic analysis of Verilog model for its structure visualization as well as to simulate the model for simulation results visualization.

The syntactically correct Verilog model is passed to the *Verilog parser* that was implemented using the ANTLR v3 parser generator [5]. Only the Verilog language constructs relevant to structural description are parsed and transformed into the internal data representation: module definition, ports definitions, instantiations of modules and their mutual connections represented by “net” construct.

[1] standardized format of simulation results

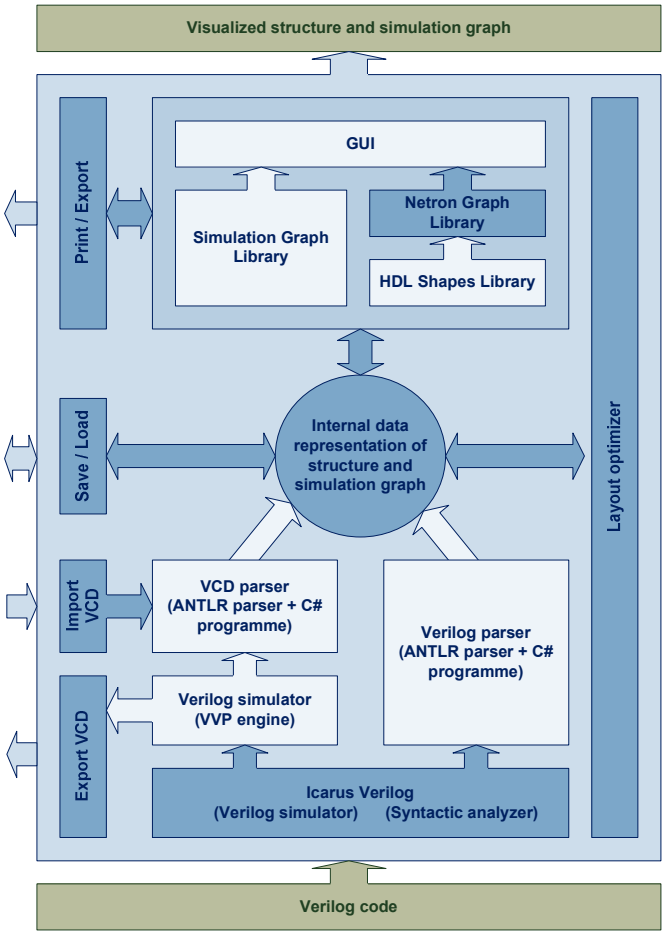

Fig. 4. The system architecture

Icarus Verilog simulator generates the simulation results in the form of VCD file – a textual description of variable values changes throughout the simulation time. A *VCD parser* is then used to perform the transformation of VCD data into the internal XML representation that was described in the previous section.

The *Layout optimizer* can be applied to the internal data to optimize the graphical layout of the visualized model. The goal of this optimization is symmetrical and well-arranged modules alignment and connections layout.

All the required application functionality is provided to the user by means of *Graphic User Interface* (GUI). To fulfill this task GUI cooperates with other components.

Simulation Graph Library component is used for drawing simulation results (Waveforms). It transforms the simulation results from internal XML form into an interactive simulation graph. The functions like zoom, time cursor, waves reordering and redrawing, image exporting or printing are supported by this library.

Netron Graph Library component is the core of the visualization. It carries our the actual drawing of modules, ports and interconnections and provides functions like objects moving, objects multi-selection, connections forming, graph printing and exporting, grid and snap to grid functions etc.

HDL Shapes Library component is an extension to Netron Graph Library. It defines the shapes of modules, connections, notes and pin labels. The basic classes for visualized objects: VerilogModule, VerilogPort and VerilogConnection are defined here. The objects of these classes represent the basic building elements of graph drawing. HDL Shapes Library is an interface between the Netron Graph Library and the internal XML representation. The classes for handling XML representation, modifying Verilog model visualization, and layout optimization are also defined here.

VI. System Implementation

The system was implemented in C# language under the .NET Framework 3.5 in Microsoft Visual Studio 2008 SP1 under the operating system Windows 7 Professional. The purpose of Verilog parser component is Verilog code parsing into the internal XML representation. It consists of three parts: VerilogLexer, VerilogParser, and VerilogToXML. The source codes of VerilogLexer and VerilogParser were generated by ANTLRv3 parser generator [5].

The VCD parser component is similar to the Verilog parser component. This component consists of three parts: VCDLexer, VCDParser, and VCDToXML. VCDLexer and VCDParser were also generated by ANTLRv3 parser generator.

HDL Shapes Library contains the definitions of classes like VerilogModule, VerilogPort and VerilogConnection that were derived from Netron Graph Library classes Shape and Connection. Other classes were implemented for handling XML representation, visualization and layout optimization.

The Simulation Graph Library provides an engine for creating interactive simulation graph (waveforms) user interface. This functionality is ensured by four main classes: Stamper, SimGraphControl, Wave and Header

VII. Results And Experience

All the implemented components have been tested during the implementation process and after that entire application was tested again. All found bugs have been fixed.

A. *Illustrating Example*

To illustrate the features of the implemented visualization tool a simple example will be presented here. In Fig. 5, there is a fragment of a simple Verilog model representing the structural description of 8-bit binary adder that was the input of the HDL Visualizator. The Verilog code is analyzed, simulated and transformed into an XML representation that is then visualized. The resulting visualization of Verilog structural description and its simulation illustrates the screenshot in Fig. 6.

B. *Restrictions*

The application is able to recognize the following Verilog language constructions: module definition with module name and ports declaration, port type definitions, all types of net definitions (supply0, supply1, tri, triand, trior, tri0, tri1, wire, wand, wor), module instantiations, UDP (User Defined Primitive) definitions and UDP instantiations. Verilog parser can handle implicit port mapping or named port mapping in the module or UDP instantiation.

The vector type port can be assigned each vector element separately using curly brackets like in .portname ({sig1, sig2,

```
...
module SUM8bit (IN1, IN2, OUT, CO);
   output [7:0] OUT;
   output CO;
   input [7:0] IN1;
   input [7:0] IN2;
   wire [6:0] carrysignal;
   SUM sum0(IN1[0],IN2[0], 0,OUT[0],carrysignal[0]);
   SUM sum1(IN1[1],IN2[1],carrysignal[0],OUT[1],carrysignal[1]);
   SUM sum2(IN1[2],IN2[2],carrysignal[1],OUT[2],carrysignal[2]);
   SUM sum3(IN1[3],IN2[3],carrysignal[2],OUT[3],carrysignal[3]);
   SUM sum4(IN1[4],IN2[4],carrysignal[3],OUT[4],carrysignal[4]);
   SUM sum5(IN1[5],IN2[5],carrysignal[4],OUT[5],carrysignal[5]);
   SUM sum6(IN1[6],IN2[6],carrysignal[5],OUT[6],carrysignal[6]);
   SUM sum7(IN1[7],IN2[7],carrysignal[6],OUT[7],CO);
endmodule
module main;
   reg [7:0] a,b;
   wire [7:0] y;
   wire co;
   time tim;
   SUM8bit s1(a,b,y,co);
   initial begin
               $dumpfile( "SUM8bit.vcd" );
               $dumpvars( 3, main );
               a = 0;
               b = 0;
               #1 a = 2;
               #1 b = 15;
               #1 a = 8;
               #1 b = 137;
               #1 a = 119;
  end
endmodule /* main */
```

Fig. 5. Fragment of Verilog structural model

sig3...})). Verilog parser doesn't support block assignment of signal arrays to the array of module instances.

VCD parser supports four state VCD file format according to the IEEE standard [18], but it does not support extended VCD file format.

C. System Scalability

The system architecture is designed with the idea of component independence in mind (each component can be replaced by a new one in case the character of interface was kept). The components are implemented as independent projects and their interfaces are well documented.

The significant space for improvements is in the Verilog parser. As it was mentioned before, the Verilog parser does not support all the Verilog language constructs and its performance is not ideal. Its performance is given by ANTLR characteristic not by an efficiency of Verilog syntax description.

For simulation purpose an external tool Icarus Verilog was used that generates the VCD file that is afterwards imported into the XML file. It is very simple and efficient solution. It has, however, one disadvantage. The VCD output is generated based on the commands specified in the input Verilog code, therefore there is a need to modify it each time we need to change the variables we would like to trace. This disadvantage is compensated by the simplicity of this solution but represents the potential space for system improvements.

The significant improvements can also be done in the visualized structure layout optimization. We could provide various optimization algorithms to get the layout as close to the users expectations as possible so that almost no manual layout editing would be necessary.

The HDL visualization tool could also be extended by a textual editor to input Verilog code. This extension would transform it into the simple development environment for Verilog projects. The application like that would be capable of writing Verilog model of digital system, checking its syntax, compiling and simulating it and visualizing its structure and simulation results.

D. Performance Tests

In order to find out the efficiency of the developed HDL visualization tool it was exposed to several performance tests. The components Verilog parser and VCD parser turned out to be the weakest parts in the chain especially when large Verilog or VCD files are to be parsed. Some experimental results for larger Verilog code compilations are given in Table I.

The experimental results prove the inefficiency of the Verilog parser. The similar problem arises over the VCD parser performance. For example to import the VCD file of 0.5MB takes about 20 seconds. This size of file corresponds approximately to the simulation time of 2000 time slices.

To conclude the tests for the reasonable processing time the input Verilog file should not exceed 1000 lines and the VCD file should not exceed 0.5 MB. The HDL visualization tool is therefore no suitable for large projects but it can meet the needs of students and beginners in Verilog design that are working on smaller projects.

VIII. Conclusion

The paper is devoted to the problem of visualization of structural digital system models described in Verilog. Some design alternatives and the possibilities of using the existing freeware tools have been discussed at the beginning. The core of the paper forms the design and implementation of the new HDL visualization tool.

The resulting application called HDL Visualization supports Verilog input and is able to visualize digital system structural description. It is also possible to simulate the design and to display the results in a waveform view. The GUI of the application is intuitive and simple. The internal representation of schematic in the XML format can be stored and later loaded and edited again. The XML file contains also the simulation results. The visualization results can be printed or exported to an image file. The developed tool can be used for smaller project and therefore is especially suitable for education purpose. However, the simultaneous visualization of structural model and its simulation is useful for verification purpose as well.

TABLE I
THE HDL VISUALIZATION TOOL EFFICIENCY IN TERMS OF VERILOG CODE PROCESSING

Verilog file size (number of lines)	Processing time (sec.)	Memory Consumption (MB)
630	4	150
1050	8	200
1995	15	400
3254	60	1500

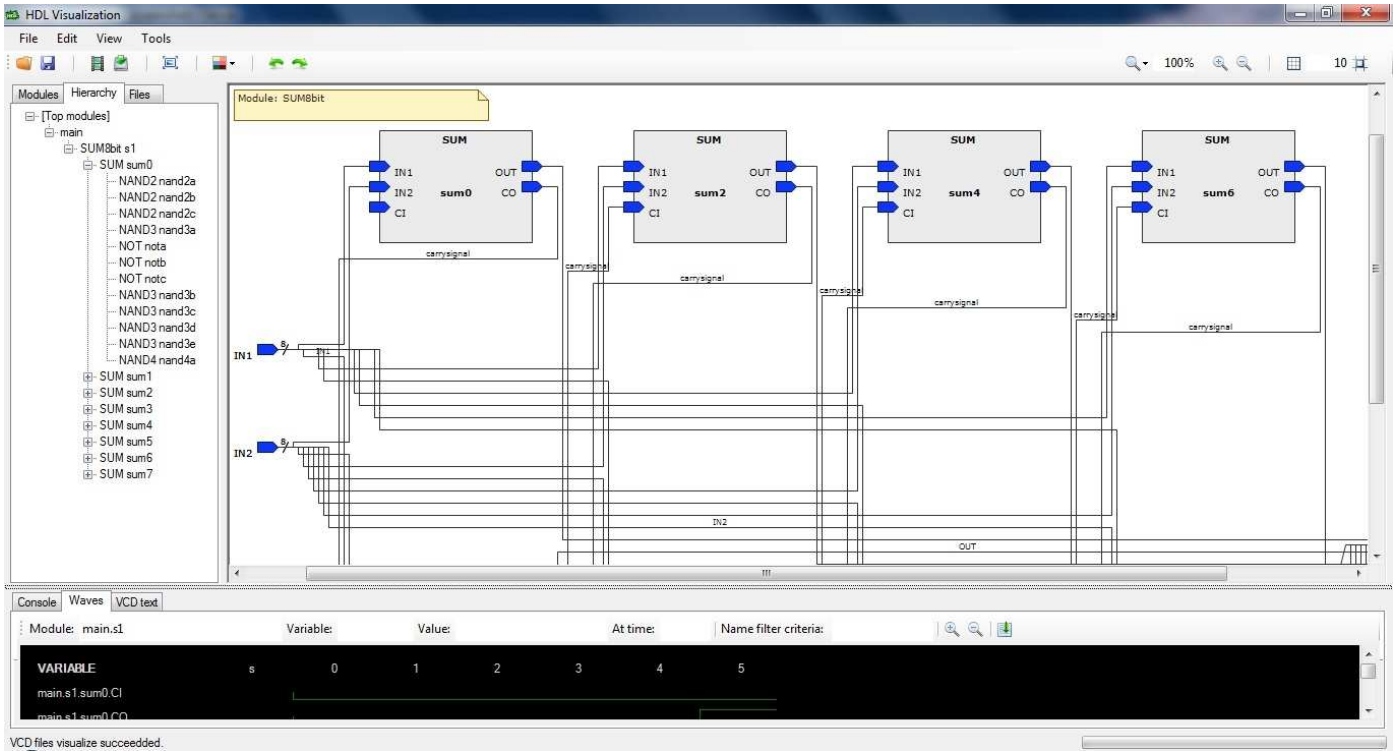

Fig. 6. Screenshot of HDL visualization tool

ACKNOWLEDGMENT

The support by Slovak Science Grant Agency (VEGA 1/0649/09 "Security and reliability in distributed computer systems and mobile computer networks") is gratefully acknowledged.

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] R.M. Volkmann, ANTLR 3, Online, January 2010. http://jnb.ociweb.com/jnb/jnbJun2008.html

[6] D., Macko, "Development of visualization environment for supporting the digital systems design," in *Proc. of the 6th Student Research Conference in Informatics and Information Technologies*, Bratislava, April 2010, pp. 419-426.

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

[8] 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.

[9] Verilog.net, "Verilog.Net Free Tools," Online, May 2009. http://www.verilog.net/free.html

[10] W. Kelly, "The Gardens point parser generator (GPPG)," Beta version 0.8, 2005, Online, May 2008. http://sharptoolbox.com/tools/gardens-point-parser-generator

[11] P. Cederberg, " GRAMMATICA – parser generator", v. 1.4, 2003. Online, May 2008. http://grammatica.percederberg.net/index.html

[12] University of Linz, "The compiler generator Coco/R," 2006. Online, May 2008. http://www.ssw.uni-linz.ac.at/Coco/

[13] J. Willis, et al. "AIRE/CE - Intermediate representation with extensibility / Common environment. Internal intermediate representation (IIR) specification," v 4.6, 1999. Online, May 2008. http://www.ececs.uc.edu/~paw/aire/iirUS.pdf

[14] M.H. Reshadi, B. Goji-Ara, Z. Navabi, "HDML: compiled VHDL in XML," in *IEEE Xplore*, 2002, ISBN: 0-7695-0890-1. Online May 2008. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=890270

[15] W. Ecker, et al., "Using xml for vhdl model representation," 2000. Online, May 2008. http://www.ifip.org/con2000/icda2000/icda-16-4.pdf

[16] GigaScale Systems Research Center, " MoML — a modeling markup language in XML," University of California at Berkeley, v. 0.4, 2000. Online, May 2008. http://www.gigascale.org/pubs/16/moml_erl_memo.pdf

[17] T. Germano, "Graph drawing," 1999. Online, May 2009 http://davis.wpi.edu/~matt/courses/graphs/

[18] IEEE Computer Society, "IEEE Standard Verilog® Hardware Description Language," ISBN 0-7381-2827-9 SS94921, 2001.