Graphical Environment tool for testbench conception

Marcelo Pereira Barros
Escola de Engenharia Elétrica, Mecânica e de Computação
Universidade Federal de Goiás
warriorfly@gmail.com

Alex Mendes Martins
Escola de Engenharia Elétrica, Mecânica e de Computação
Universidade Federal de Goiás
alexengcomp@gmail.com

Karina Rocha G. da Silva
Escola de Engenharia Elétrica, Mecânica e de Computação
Universidade Federal de Goiás
karinarg@emc.ufg.br

Cássio Leonardo Rodrigues
Instituto de Informática
Universidade Federal de Goiás
cassio@inf.ufg.br

Adriano Cesar Santana
Escola de Engenharia Elétrica, Mecânica e de Computação
Universidade Federal de Goiás
Adriano@emc.ufg.br

ABSTRACT

Functional verification is a very difficult part of the entire design. It spends almost 70% of the resources. For this reason the engineers are supposed to use all the necessary tools in order to decrease the verification bugs and costs. This paper presents the creation of an graphical auxiliary tool to design the blocks that will compose the design, in order to generate the testbench to make the functional verification.

Keywords
Functional Verification, VeriSC, ETBC

1. INTRODUCTION

Functional verification is a technique to demonstrate that the intent of a hardware design is preserved in its implementation [01;02;04]. In fact, there is no consensual functional verification methodology in the digital circuit industry. Each design house tailors its methodology according to the type of digital circuit to be produced, the resources that are available and constraints that are imposed by the project. However most functional verification methodologies comprise four basic components: i) the Register Transfer Level (RTL) design under verification (DUV); ii) a set of specifications that the design must comply with; iii) a simulation mechanism to judge the DUV against its set of specifications; iv) a mechanism to estimate the level of confidence achieved during the functional verification process. Except for the DUV, the remaining components are enclosed in an environment called testbench.

The testbenches implementation can take a considerable amount of time in the verification process. Some reasons, which can increase the time for testbench implementation, are the number of connections between modules/blocks, the time spent to adapt the testbench to the Design Under Verification (DUV) and the number of module instances, transaction data structures and transaction communication channels. A methodology that makes the complex verification process easier and a tool to implement this methodology and generate automatically testbench prototypes can be a good approach to reduce the overall time of project flow. This paper presents a tool to graphically design the block of modules that compose the hierarchical DUV. These design is used to generate a TLN file which is used to input the eTBC tool [03]. The eTBc (Easy TestBench Creator) is a semi-automatic testbench generation tool, which has been used to generate testbench prototypes used in the VeriSC methodology [01], as explained in the next sections. VeriSC methodology guides the implementation of working testbenches during hierarchical decomposition and refinement of the design, even before the RTL implementation starts. More information can be obtained in [01].

2. EASY TESTBENCH CREATOR

The Easy Testbench Creator (eTBc) is a tool for semiautomatic testbench generation. This tool works as a code generator, receiving two files as input: A Transaction Level Netlist (TLN) file described by the functional verification engineer (user of eTBc). This file is a model of the IP-core using Transaction data and RTL data. The RTL data used in this level is only the name of modules I/O ports. The remaining of the description is only transaction data. The language used in this level is eTBc Design Language (eDL). eDL is a simple language used to describe modules, connections, FIFOs and some data in the RTL level.

The another input file is a template of a testbench element that will be generated. The eTBc template is a way that eTBc works to generate testbench elements. The role of templates is to guide eTBc to generate code, based in one TLN.

The TLN defines the model of system and the template defines the model of testbench. The templates are created using eTBc Template Language (eTL) and there are implemented templates in VeriSC methodology for SystemC and Verilog languages. eTL is a language that allows adjustment to a specific methodology and HDL. If a Verilog/SystemVerilog, VHDL or SystemC specific methodology wants to generate testbenches in a specific way, a template can be written using eTL for this purpose.

SForum 2012 - Student Forum on Microelectronics
This work has been developed by the first author(s) in the scope of the undergraduate studies

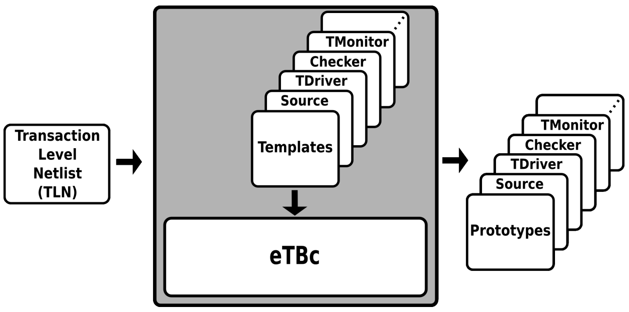


Figure 1: Architectural Model

```
struct coeffs {
    trans {
          short coeff [64];
    }
    signals{
          signed[8] in_pqf;
          bool valid;
          bool ready;
    }
}
module PIACDC {
    input coeffs pi_in;
    output coeffs pi_out;
}
module QI {
    input coeffs qi_in;
    output coeffs qi_out;
}
module P_mpeg {
    input coeffs m_in;
    output coeffs m_out;
    fifo coeffs pi_qi;
    PIACDC piacdc_i(.pi_in(m_in),.pi_out(pi_qi));
    QI qi_i (.qi_in(pi_qi),.qi_out(m_out));
}
```

Figure 2: Source Code written in eDL

As can be seen in the architectural model of the tool presented in Figure 1, the models are created using the eTBc Design Language (eDL). Internally the tool uses a code generator and two translators, one to interpret the TLN code and the other to interpret the code of the templates. The eDL language is used to write the TLN code to describe the model. The eTL language is used to write the templates used by the tool. If one wants to port the tool for a specific methodology, he has to create new templates according with this methodology. The eTBc can be adapted to other methodologies and languages using the template language (eTL). In this case, the verification manager/team has to implement your own templates to use with a specific methodology. After that, in order to finish the testbench, the user need to fill the ”source” module of VeriSC with stimuli generation and ”checker” with all asserts constraints according with the verification plain.

A simple example of TLN written in eDL is shown in Figure 2. This TLN is a model of an 8-bit adder. From this example shown in Figure 2, eTBc can generate all testbench elements of the VeriSC methodology discussed in section 2. This model written in eDL is not restricted only to VeriSC methodology. It can be used in any verification methodology. To use this model with another methodology is necessary other templates written in eTBc Template Language (eTL).

The problem in order to use eTBc is to learn how to implements the eDL language. Then, a more intuitive form to implements the TLN code is to design the entire module that compose the DUV and automatically infer the TLN code. This is the objective of the tool presented in next section.

3. THE GRAPHICAL TOOL

Using the eDL graphical generation tool is possible to create modules, and each module represents the hierarchical division of the design. The modules can be manually designed and linked according to the communication interface from the design.

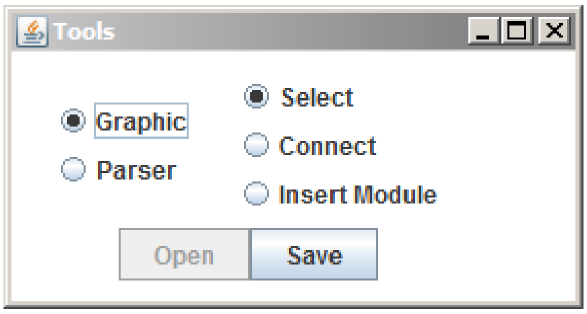
Source Code 2: File template_example.txt

```
$$(file) $$("textfile.txt")
-- This is an example of eTL language (
   VHDL comment)
// Put here any HDL code  (Verilog
   Comment)
/* SystemC comment */



$$(foreach) $$(module.in)
  $$(i.type) $$(i.name);
$$(endfor)

$$(foreach) $$(module.out)
  $$(i.type) $$(i.name);
$$(endfor)

$$(endfile)
```

Figure 3: Source Code written in eTL

The modules and their interfaces are presented on screen, making the creation and viewing more intuitive. The visual command makes the design implementation easier, faster and more intuitive and allows for better code maintenance. The user can create the blocks and the interface communication between the modules. Then, it is necessary to specify what kind of interface will be linking the modules. After that step, the visual blocks are converted to eDL code. And the eDL code can be used to generate the testbench design.

The application allows also the opposite way, generating a visual presentation of the project by inserting code EDL. The software includes a Tool Window (Figure 4) that allows you to choose the form that it will work.



[✓ Graphic] Parser

[✓ Select] Connect Insert Module

Figure 4: Tools Window

3.1 Graphic Mode

The graphic mode application can be used selecting Graphic in the Tools Window. In graphic mode, the Main Window will be released for editing, in which the user will interact to graphically building the DUV. The eDL source code resulted will be generated automatically in real time, on the Parser window, which is locked for editing.

The Window allows choosing three types of iteration to the Main Window: insert module, connect, and select.

3.1.1 Insert Module

The insert module option allows to insert blocks in the window, simply by pressing the mouse button, drag to define the size, and drop. These blocks represent a hierarchical module from the DUV. Using the example shown in Figure 2, piacdc_i and qi_i are instances of modules of PIACDC and QI, respectively, and are instantiated on a parent module, in this case P_mpeg.

When inserting a block, a module is created, and the same is instantiated in the parent module, which is the one who first pressed the mouse button.

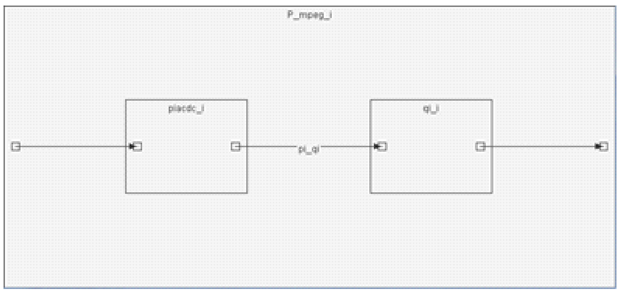


Figure 5: Main window

3.1.2 Connect

The connect option, allows the user to connect two blocks with arrows by clicking and dragging on a block to another. The user can connect two children blocks, a parent block to a child block or otherwise.

The connection between the blocks represents the form of communication between the modules. Again, the example of Figure 2 shows that there is an output instance of piacdc_i (pi_out) connected to an input instance of qi_i (qi_in) through a fifo connection (pi_qi). These interfaces, input, output and fifo, represent instances of the structure coeffs.

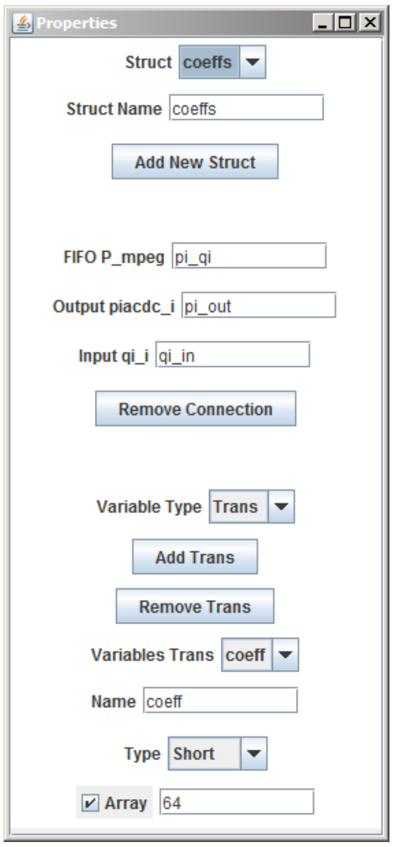
By clicking on a child block and drag it to another, a new structure is created and instantiated in three different places: in the parent module, as a fifo, and both children, as input and output. Then, the blocks are connected by inserting this information in the instance of both modules.

A similar procedure occurs if you connect a parent block to a child block, except for the absence of instance fifo, which is not necessary.

For connections of instances of modules, you can also use the same output structure connecting the various inputs. Note that after you insert an arrow, are created in addition to the arrow, two small boxes at the ends. These boxes, which represent the input and output structures, may be used to make new connections, reusing instances of their structures.

3.1.3 Select

In the select option, the user can be free to interact with graphical objects created by resizing or moving the blocks. The user can also change the characteristics of an object by double clicking. This opens a properties window (Figures 6-7) with specific characteristics of the object.



Struct coeffs
Struct Name coeffs
FIFO P_mpeg pi_qi
Output piacdc_i pi_out
Input qi_i qi_in
Variable Type Trans
Variables Trans coeff
Name coeff
Type Short
✓ Array 64

Figure 6: Properties Window – Connection

The block properties will display the name of the module that is instantiated and the instance of the module. For the given example, a double click on the block piacdc_i shows the Figure 7. Names can be changed and the module instance (the block) may be removed by clicking Remove Instance Module.

The connection properties (Figure 7) will show the name of the implemented structure and interfaces (input, output and fifo). It is this window which will be also included variables on the structure, by first selecting the type of variable (trans or signal), and then adding, removing, renaming or changing the variables.

The same structure can be used in multiple connections. The Figure 2 shows that all interfaces are instances of the same structure, called coeffs. Thus, the graphic implementation, after performed the connections, just modify one of them to represent the structure coeffs, inserting and renaming the variables. For other connections, just change the first box called Struct, in the Properties Window (Figure 8), to reference the same structure coeffs. The structures which do not appear in any connection is automatically rejected. This is also valid for modules, which can have more than one declared instance.

Finally, the Tools Window still shows the Save button to export the EDL generated code directly to a file.

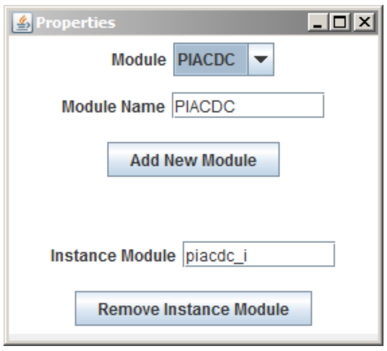


Figure 7: Properties Window - Instance Module

3.2 Parser Mode

The software still has an inverse way that meets the requirements of compatibility and standardization. Simply selecting the Parser in the Tools Window (Figure 4). So the Main Window (Figure 5) is locked for editing by releasing the Parser window. Changes made to the inserted code automatically reflect the Main Window, unless you find a bug in the inserted code. Errors are lexical, syntactic and semantic checked.

The user can still click the Open button in the tool window to select a file with saved eDL code.

4. Results

Using the Graphical Environment (GET) tool in order to generate the testbench provide some advantages when compared with making it manually. One big advantage is not having to learn how to build the TLN file. The other advantage is the number of line codes that can be saved using the tool. The engineer does not have to generate any code lines, when using the GET tool.

The next table uses code from an MPEG4 decoder design, in order to make the comparison between number of code lines a design using the TLN file and using GET tool. This MPEG4 decoder design is part of Brazil-IP project [7].

Design	Lines of TLN	Lines with GET
MPEG4	25	0

5. CONCLUSIONS

This paper presented a graphical auxiliary tool to design the blocks and interfaces that will compose the DUV to be implemented. The tool can be used to generate the testbench. With this tool the engineer do not need to be concerned with any specific code. He only needs to make a design of his hierarchical DUV and the tool will generate the TLN to be used in the eTBc tool.

6. ACKNOWLEDGMENTS

This work was supported by grants from the CNPq sponsor agency.

7. REFERENCES

[1] K. R. G. da Silva, E. U. K. Melcher, I. Maia, and H. do N. Cunha. “A methodology aimed at better integration of functional verification and rtl design”, Design Automation for Embedded Systems, Volume 10, Number 4, pp. 285-298, December 2005

[2] J. Bergeron, editor. Writing Testbenches. Springer, Boston, 2003.

[3] Maia, I., Silva, K. R. G., Max, L., Camara, R., & Melcher, E. U. K. (2007). eTBc: A Semi-Automatic Testbench Generation Tool. IPSOC (pp. 1-5) .

[4] O. Lachish, E. Marcus, S. Ur, and A. Ziv. Hole analysis for functional coverage data, 2002.

[5] R. Grinwald, E. Harel, M. Orgad, S. Ur, and A. Ziv. “User defined coverage: a tool supported methodology for design verification”, DAC ’98, pp. 158–163, New York, 1998.

[6] BrazilIp. www.brazilip.org.br/fenix. 2007.

[7] A. K. Rocha, P. Lira, Y. Y. Ju, E. Barros, E. Melcher, and G. Araujo. “Silicon validated ip cores designed by the brazil-ip network”, IP/SOC 2006, June 2006.Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825.