FUNCTIONAL VERIFICATION OF VERILOG DESIGN USING A VERISC METHODOLOGY TESTBENCH APPROACH

Adriana Ferreira de Brito, Raphael de Souza Neves, Karina Rocha Gomes da Silva
Escola de Engenharia Elétrica, Mecânica e de Computação
Universidade Federal de Goiás (UFG)

ABSTRACT

Functional verification is a very difficult part of the entire chip design. It spends almost 70% of the resources. The focus of this work is to propose a functional verification methodology for Verilog designs, using the VeriSC methodology [1] as a basis, and using the Verilog language to implement the testbench. With this methodology will be possible to design a testbench that can identify functional errors as soon as possible in the Design Under Verification (DUV).

1. INTRODUCTION

Verification aims to ensure that the results of a design is consistent with the expected. One way to make a verification process more efficient would be automating it, making the process faster and more predictable [2]. The testbench is an option to conduct the verification in order to focus on identify the errors as soon as possible, because if the errors propagate to the other phases from the design, it became more difficult to catch them. The testbench is an environment in which the device to be verified (Design Under Verification – DUV) receive the specified stimuli and it results can be compared with the ideal results, coming from the reference model. With VeriSC methodology the implementation of the testbench is performed before to start the development of the DUV, together with the development of the reference model [1].

There are three types of verification: static, dynamic and hybrid, the dynamic verification is known as functional verification "it is the default strategy of the CIs industry" [3]. Functional Verification has a testbench, that is an environment composed of reference model (ideal model), implementing all specified features of device [1] and the design Under verification (DUV). The testbench is responsible for the stimuli generation. It also have the output stimuli from the DUV and compares the Reference Model output with the output coming from the DUV [4]. The testbench generated stimuli grow to the size and complexity of the device, then is almost not possible to make an exhaustive simulation. Because of this, one can uses constrained random stimuli, with some kind of coverage to control the randomicity and the parts of the project that has been verified [1].

There are many methodologies to build a testbench [5], for example, it is possible to write the linear testbench where each entry has its continuously variable and constant input stimuli, resulting in an exponential increase of stimuli in accordance with the increased number of sets. It can become impossible the representation of all combination of input [6]. There is a methodology to build the testbench using input/output files, the testbench will have an interface between the input/output files and the DUV [7]. The testbench can be made using state machine to generate the input vector, each entry will be a state [8], or the testbench can be build using tasks and functions, each task or functions conducts a functionality, such as task of writing a file which receive the data and the address like parameter [9]. Between the methodologies the linear is the easier and simple to write but it is dependent of the complexity of the project. The testbench that uses I/O files has to manipulate archives, the state machine methodology does not support the building of a robust testbench that is necessary in complex projects. Testbench using tasks and functions are more efficient in devices that perform calculations. [6, 7, 8 and 9].

2. VERISC METHODOLOGY

The functional verification methodology underlying our work is VeriSC [1]. The VeriSC methodology allows the generation of the complete running testbench before the implementation of the DUV has been started without requiring extra code to be written. Hence, the design can be verified in all necessary phases of its implementation, mainly at the beginning of the DUV implementation. Furthermore, the VeriSC methodology can reuse its own elements to implement the testbenches, to perform a self-test to reveal errors in the testbenches. [1], leaving the testbench and simulation done "before the development of DUV" [4]. In the new verification flow proposed for the VeriSC it is implemented a mechanism that simulates, with elements of the testbench, the presence of DUV [1]. In VeriSC the testbench is implemented with the following blocks: Source, TDriver, TMonitor, Reference Model and Checker. The stimuli for simulation are generated by the Source, the Source is linked to Reference Model and to TDriver. TDriver is an interface that connect the Source to DUV and does the communication protocol between these blocks. The output of DUV is linked to TMonitor, TMonitor is the interface between DUV and Checker and does the communication protocol between these blocks. The Reference Model output is also connected to the Checker that performs the comparison between the two outputs. To

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

develop the testbench is used the SystemC hardware description language [1].

3. VERIFICATION WITH VERISC USING VERILOG

After the verification, the next step in the design flow is to synthesize the DUV in order to have a netlist. Using VeriSC with language SystemC is necessary to translate the SystemC language to languages like Verilog or VHDL [10]. In order to avoid the use the intermediate tools is better to build a methodology that have as a base the main idea of the VeriSC methodology but with the development of the DUV using Verilog language. This methodology inherits from VeriSC Methodology the main idea of building testbench before the DUV, and all the schema from testbench modules, like TDriver, TMonitor, Reference Model and Checker, but the stimulus are saved in archives with test vector, as can be seen in Figure 1.

The Source generates high level stimuli and stores them. Each generated value will be one input vector of the Reference Model. A C program converts them to low abstraction level. The TDriver module, will read the file and one output stimuli will be assigned the output port, between the TDriver and the DUV. Then, the communication protocol (handshake) will ensure no loss of stimulation between the modules

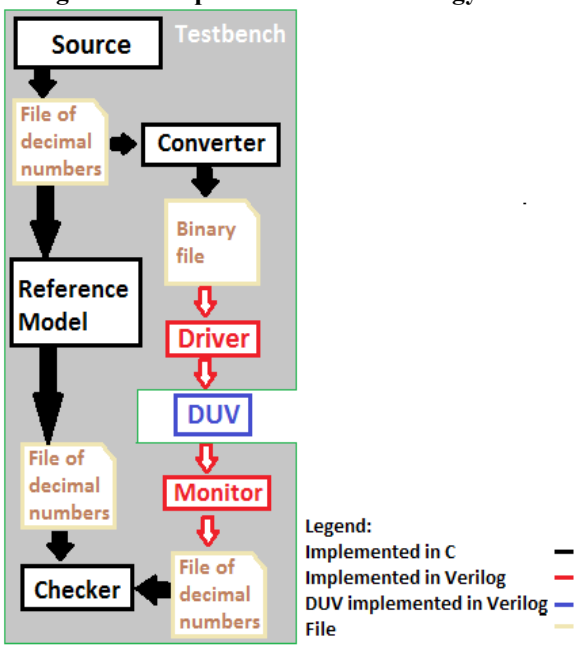
The Reference Model works in transaction-level data. It makes it more simple to be implemented. After the execution of the Reference Model with the same stimuli as the DUV, the output is recorded in another file, each line is an output. The DUV receives input stimuli from the TDriver, it is run and stimuli are obtained from the output ports, the stimuli are stored in another file. The TMonitor module is the interface where the output stimuli are stored in the decimal file, between the DUV and the TMonitor there is the communication protocol that ensures the data integrity.

The testbench performs the comparison of the output file generated by the Reference Model and the output file generated for DUV, line by line comparing the two files. If it finds any divergence between the outputs, it will show which line is divergent in the archives, thus it is possible to trace which input, or set of entries, that generated this output stimulus that is divergent in the DUV file and the Reference Model file. The TDriver, TMonitor and TDriver are implemented in Verilog.

For the standardization of nomenclature and data aims to create a file in which the inputs and outputs of each program are identified.

The proposed methodology schema is shown in Figure 1.

Figure 1: Template of the Methodology



4. RESULTS

Using this methodology almost all the modules are ready to use before the DUV is ready. The Source and the Converter modules are almost ready, only needing to set the stimulus that will be generated: number and size of the data, because this information may vary depending on the design. The Reference Model depends on the device design. The Checker can be fully reused because it only carries out the comparison of line by line and warning if the difference between files.

This methodology tested some basic devices like adder, multiplexer, state machine and dpcm, with these devices was possible to identify changes that are necessary according to each project: size of ports, size of data, number of ports (stimuli), connection between the TDriver ports and the DUV ports, connection between the DUV ports and the TMonitor ports and Reference Model. Using this methodology the Checker code is one hundred percent done, it is not necessary to change anything. Tests were also made using the labs (1 to 7) proposed by Altera [11].

The main test was carried out using the methodology to implement and verify a device to alert the parents that they forgot the child in the car, named Smart Car Baby Seat.

4.1. Adder

Adder device is as simple as the state machine, this device can verify the correct functioning of the testbench using VeriSC with Verilog. The methodology was implemented to generate a data file from the input Source program, a program was developed for converting decimal numbers to binary numbers, implemented the Reference Model and the Checker, all developed in C

language. Used the TDriver and the TMonitor to be the interface between the DUV and archives, both was implemented in Verilog. The output file DUV was the same as the output Reference Model, confirmation was obtained by Checker.

4.2 DPCM

The methodology VeriSC with Verilog was tested in a Differential pulse-code modulation (DPCM) design. It is a device that receives one data and guard in a buffer until it receives the next. Comparing with the adder design the DPCM is more complex, it has input data (stimuli) of different size, and it uses a internal state machine. Even though it is more complex than adder, using the methodology, it was easy.

In the Source, Converter, TDriver and TMonitor was changed the number of stimuli (data) and the length each one. Reference Model was development for represent the features of DUV. The Checker not changed anything.

In these devices were used the handshake, communication protocol, between the TDriver – DUV and DUV – TMonitor, thus was added the input and output in TDriver, DUV and TMonitor to do this communication. The module was changed for support this protocol, ensuring the integrity of data.

4.3 Smart Car Baby Seat

It was proposed the build a device that would ensure that the parents or guardians do not forget the kids in the car baby seat. It works with five sensors: three in the baby car seat, one in the driver's door and one in the ignition. Like output, warning of dangers, will be two leds: one warning that have baby inside the car, and the door is opened or the key is in the ignition; the other alert is when the two leds are connected, it warns that have baby inside the car and the car is closed, probably with no adult in, because do not have key in ignition and the door is closed.

The development of device was with six input ports (five sensors and one clock), two output ports and four ports for handshake. The methodology VeriSC with Verilog was used in this design, as the input stimuli are only one bit for do the Source, it was necessary to change a few lines in the code, just as in the TDriver and TMonitor. The Reference Model is basically logic operation between the seat sensors with ignition sensor and driver's door sensor. The Checker is ready, have only run the program for compare the two files, one file generated by the DUV and other by the Reference Model.

5. CONCLUSIONS

The methodology proved effective with Verilog for functional verification, incoming stimuli and stimuli in the output files are saved and you can trace each entry, or set the input, for each output, or output set. The codes are reusable, it is necessary some modifications according the design in the Source, Converter, TDriver and TMonitor. The Checker is the only that do not have changes to do, and the Reference Model is the unique that have to be totally modified. With reusable code the engineer can save time on the project because the verification will be faster and more efficient.

Using Verilog in the implementation of the TDriver and TMonitor makes it possible to implement the DUV in Verilog, eliminating the translation of a language to Verilog and avoiding possible mistakes in the DUV code.

6. ACKNOWLEDGMENTS

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

7. REFERENCES

[1] K. R. G. da Silva, “Uma Metodologia de Verificação Funcional para Circuitos Digitais”, Paraíba, Brasil, 2007.

[2] J. Bergeron, “Writing Testbenches using SystemVerilog”, Springer Science, New York, USA, pp 2-21, 2006.

[3] E. L. R. Tobar, “Contribuições à Verificação Funcional Ajustada Por Cobertura Para Núcleos de Hardware de Comunicação e Multimídia”, São Paulo, Brasil, 2010.

[4] E. Melcher, “Verificação funcional Curso do programa Brazil-IP”, http://lad.dsc.ufcg.edu.br, 2012.

[5] Introduction, http://www.testbench.in/TB_01_INTRODUCTION.html, 2012.

[6] Linear TB, http://www.testbench.in/TB_02_LINEAR_TB.html, 2012.

[7] FILE I/O TB, http://www.testbench.in/TB_03_FILE_IO_TB.html , 2012.

[8] STATE MACHINE BASED TB, http://www.testbench.in/TB_04_STATE_MACHINE_BASED_TB.html , 2012.

[9] TASK BASED TB, http://www.testbench.in/TB_05_TASK_BASED_TB.html , 2012.

[10] W. Muller, W. Rosenstiel and J. Ruf, “SystemC Methodologies and Applications”, Kluwer Academic Publishers, Boston, USA, pp 217-245, 2003.

[11] Digital Logic - Laboratory Exercises, http://www.altera.com/education/univ/materials/digital_logic/labs/unv-labs.html , 2012.