An Assertion Library for On-Chip White-Box Verification at Run-Time

José Augusto Nacif[1], Flávio Miana de Paula[2], Harry Foster[4], Claudionor N. Coelho Jr.[1], Fernando Cortez Sica[1,5], Diogenes Cecílio da Silva Jr.[3], Antônio Otávio Fernandes[1]

[1] *Computer Science Department*, Universidade Federal de Minas Gerais, Brazil
[2] *Mindspeed Technologies ™*, USA
[3] *Electrical Engineering Department*, Universidade Federal de Minas Gerais, Brazil
[4] Verplex Systems, USA
[5] Computer Science Department, Universidade Federal de Ouro Preto, Brazil

Abstract

White-Box Verification is a technique that locates a failure during a design simulation without the need to propagate the failure to the I/O pins. Using assertions in the White-Box Verification provides the mechanism needed to locate the failure. This technique has gained strength in the past few years, have been used in simulation, emulation, formal verification and semi-formal verification. In this paper, we present a novel technique that extends the usage of White-Box Verification to the released product. We call this technique run-time verification with assertions. We present the results of creating a library for run-time verification, and show that the overhead of this library is relatively small for an overall FPGA chip.

1: Introduction

In the past few years, with the improvement of synthesis techniques, RTL-based design from hardware description languages has settled as the designer's choice for design entry and methodology.

RTL-based designs have enabled more aggressive design validation techniques based on *White-Box Verification* as opposed to *Black-Box Verification*. *Black-Box Verification* refers to the approach of providing stimulus to the input pins of a design and checking the results in its output pins. This approach offers very poor observability and controllability since a failure inside the design has to propagate to the output pins to be observable. Further, the failure may be noticed several hundreds of cycles after it actually happened, making it difficult to reproduce the failure.

White-Box verification is a technique to improve observability of a design by embedding assertion monitor, which we call assertions in this paper. Using *White Box Verification*, a designer can locate a failure internal to the design because assertions can trigger immediately after an error occurs.

Assertions are inserted into a design based on the knowledge about legal and illegal behavior of internal design structures [1][2]. Usually, the assertions are inferred by a designer according to interface rules or unwanted corner cases of the design.

Assertions can be built from hardware description languages[3], from some pragmas of a specific tool such as [4], or from a testbench written using a testbench language, such as OpenVera [5]. *White-box verification* has become a popular design validation technique, improving the confidence level in a design because assertion monitors, acting like probes inserted into a chip, solve the observability problem of testing chip designs [6][7].

To the best of our knowledge, *White-Box Verification* has only been applied during the simulation, formal analysis and emulation[8][9][10][11] phases of a design. Its use has been limited to the validation phase because of its initial goals, which were to capture the design bugs as the design progresses. However, because of the complexity of current designs, exhaustive test or verification can not be accomplished. As a result, using *White-Box Verification* does not guarantee a bug-free design.

In the context of chip-level designs implemented in field-programmable gate arrays (FPGAs), assertion monitors enable the monitoring of reconfigurable designs at run-time after deployment of the design. If a bug is found in the design, an assertion engine stores the error information and later notifies the designer. Because the error information is directly linked to an RTL design, the designer can more easily locate the problem, and is then able to provide a new version in a very short time.

In this paper, we propose an assertion engine architecture to be used in reconfigurable designs by extending the use of the *White-Box Verification* technique beyond the simulation/emulation phases of a design. This technique is based on the *Boundary-scan* [12] concept, along with synthesizable assertions. We suggest that the overhead caused by these synthesizable assertions is minimal and all design assertions can be implemented in the design. Therefore, without compromising power, area and package costs one can debug a taped-out design in run-time during its normal operation.

This paper is outlined as follows. We describe the use of assertion libraries as an engine in Section 2. We discuss work related to ours in Section 3. In Section 4, we present an implementation of assertions for run-time debugging. In Section 5, we present the results. Finally, in Section 6 we conclude with our remarks and present future work.

2: White-Box Verification Engine

White-box verification is based on the notion of adding monitors to a design described by a hardware description language. White-box verification adds greater flexibility to hardware design validation, because it allows designers to observe unwanted conditions specified by the assertions. During simulation, as soon as an assertion fails, the simulation stops. In this paper, we are interested in assertions that are actually used in the design process, and more specifically assertions that can be synthesized. As a result, we use the assertions from the Open Verification Library (OVL)[3].

Table 1 lists the assertions defined from the OVL. Assertions can be classified as either deterministic or non-deterministic. Deterministic assertions will trigger after a fixed number of cycles if a design failure is detected. On the other hand, non-deterministic assertions will be tested only after an initial (or triggering) event occurs, and as a result, may never be tested (if the triggering event never occurs).

3: Related Work

The existing work on assertions has focused mainly on white-box verification for simulation, formal verification or semi-formal verification [4][13][14][15][16][17].

To the best of our knowledge, the only published work related to using synthesizable assertions as a tool to support debugging a design beyond the simulation phase is presented as *Assertion Processor* in [8].

The *Assertion Processor* has been leveraged from Axis ReConfigurable Computing (RCC) technology. By definition, the RCC runs all assertions at the hardware speed. Whenever an assertion fires, the RCC generates an interrupt to a software component, which then processes the assertions using behavioral code such as PLI calls.

Although this approach is powerful, it is focused on bringing the assertion-based verification to the emulation phase of a design.

Deterministic Assertions	*Non-Deterministic Assertions*
Assert_always	Assert_change
Assert_always_on_edge	Assert_cycle_sequence
assert_decrement	Assert_frame
assert_delta	Assert_handshake
assert_even_parity	Assert_next
assert_implication	Assert_range
assert_increment	Assert_time
assert_never	Assert_unchanged
assert_no_overflow	Assert_width
assert_no_underflow	Assert_win_change
assert_odd_parity	Assert_win_unchange
Assert_one_cold	Assert_window
assert_one_hot	
assert_proposition	
Assert_quiescent_state	
Assert_transition	
Assert_zero_one_hot	

Table 1. OVL assertions by category

4: Implementation of assertion for run-time debugging

The basic idea of an assertion engine targeted at run-time debugging of a live design is based on the IEEE-1149 standard [12], which defines test logic that can be included in an integrated circuit for testing.

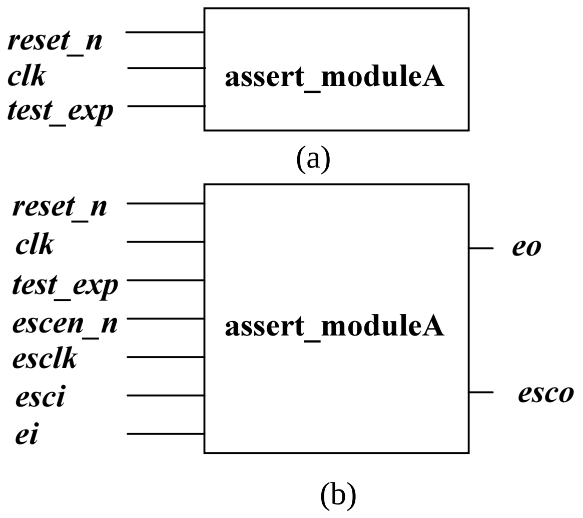

Figure 1. a) Typical OVL assertion; b) OVL assertion modified for *scan-chain* architecture

In Figure 1.a, we present a typical assertion module from the OVL. In order to use it in a *scan-chain* architecture we need to define extra pins, three inputs and two outputs, as shown in Figure 1.b. Table 2. contains descriptions of each signal.

Signal	*Description*	*I/O*
reset_n	Reset active low	Input
clk	System clock	Input
test_expr	Any HDL test expression	Input
escen	Error Scan Enable	Input
esclk	Error clock	Input
esci	Error Scan Input	Input
esco	Error Scan Output	Output
eo	Error Output	Output
ei	Error Input	Input

Table 2. Signal descriptions for Figure 1.b

The run-time version of the assertion library is extended with signals *eo, esco, ei, esci, esclk* and *escen*. Signal *eo* stands for the error output, which is the conjunction of all possible errors, and it signals that an internal error triggered by a failed assertion has occurred inside the chip. Signal e*scen* disables the test expression evaluation in the assertion, enabling the scan-chain to be exercised. Signals *ei, esci, esco* and *esclk* are error in, normal scan input, output, and clock signals.

A chip's interface will be enlarged with signals *eo, esco, escen* and *esclk.* Figure 3 presents a typical timing diagram showing how to debug a chip after an assertion fails in our assertion engine. After the chip lowers the error output signal *eo*, a monitor starts investigating which assertion has triggered *eo* by first enabling the error scan chain (by asserting *escen*). At this time, the first assertion result appears in the error scan output (*esco*). Then, at subsequent cycles, by pulsing *esclk*, a new assertion result appears at *esco* until all assertions have been scanned. For a design with *n* assertions, after *n* cycles, all assertion violations will appear as zeros in the output of *esco*.

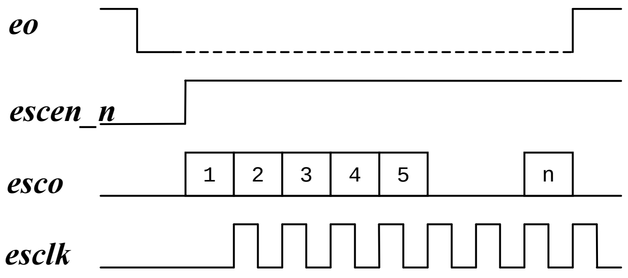

Figure 3. Behavior of the eo, escen, esco and esclk pins

In the next figure (Figure 4), we present how the design hierarchy can be chained in the design to be used by an assertion engine.

In this figure, each dark circle represents an assertion instantiation, and each clear circle represents a module. As one can see, each *esci* input of one assertion is linked to the *esco* output from the previous assertion. It is important to note that for proper debugging functionality, the designer must know the chain order of the assertions.

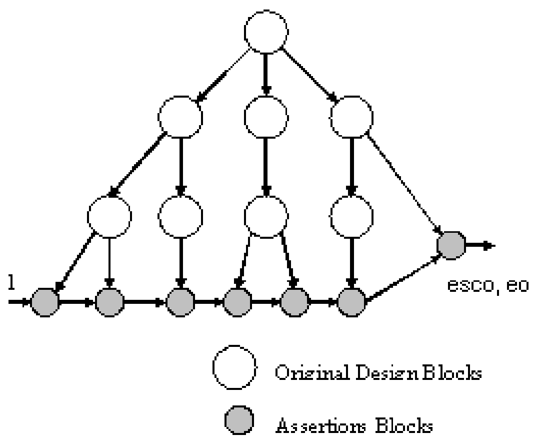

Figure 4. Design hierarchy with the addition of assertion blocks

By adding these assertion blocks to the original design, it is possible to detect an error in real time, even after testing stages.

It is important to note that deterministic assertions behave differently than non-deterministic ones for debugging purposes. Because timing information is precise in deterministic assertions, they trigger at the exact moment when a bug is observed in the design. Non-deterministic assertions, on the other hand, do not trigger at the exact time when a problem occurs because the assertion failure can occur several cycles after a triggering event has enabled the assertion.

5: Results

We implemented a representative number of assertions to obtain a run-time debug version of OVL. The synthesis was made using FPGA Express Tool for a Xilinx XCV 300E FPGA. Table 3 shows the area used by assertion modules.

The area used by assertions varies from 3 LUTs to 149 LUTs depending on the type of the assertion and how many bits are being asserted. The FPGA used has a total of 6,912 LUTs, so 3 and 149 LUTs are 0.04% and 0.46%.

Assertions	*LUTs*		*FFs*	
	Min	*Max*	*Min*	*Max*
assert_always	3	3	1	1
Assert_never	3	3	1	1
assert_add_parity	3	13	1	1
assert_range	3	15	1	1
assert_decrement	12	76	5	33
assert_increment	12	76	5	33
Assert_proposition	3	3	2	2
Assert_transition	5	53	2	2
assert_no_transition	5	53	2	2
assert_delta	4	131	3	33
assert_window	5	5	2	2
assert_underflow	4	15	2	2
assert_overflow	4	15	2	2
assert_one_hot	3	107	1	1
assert_win_change	8	54	4	35
assert_win_unchanged	8	70	4	35
assert_time	79	79	34	34
assert_change	87	133	36	67
assert_unchanged	87	149	36	67

Table 3. Area used by assertions

6: Conclusions

White-Box Verification has been used as a better choice than *Black-Box Verification* for validating designs. However, the *White-Box Verification* has only been used for the simulation and emulation phases of a design.

In this paper, we described a technique to extend the *White-Box Verification* to the final product into what we call assertion engine for *White-Box Verification*. We presented the results of creating a library of assertions for run-time verification. This library adds minimal overhead to any design. In the future, we will develop a tool that adds run-time verification capability to an existing design, and we intend to apply this design technique to several designs.

7: Acknowledgments

This paper is partially supported by CNPq.

8: References

[1] Bergeron, J., “Writing Testbenches Functional Verification of HDL Models”, Kluwer Academic Publishers, 2000.

[2] 0-In Design Automation, Inc. “Assertion-Based Verification for Complex Designs”, The Verification Monitor, January 2002.

[3] www.verificationlib.org

[4] 0-In Design Automation, Inc., "Black & White Assertion-Based Verification Flow", The Verification Monitor, May 2002.

[5] Synopsys, Inc., "Assertion-Based Verification", May 2002.

[6]Gupta, A. “Assertion-Based Verification Turns the Corner”, IEEE Design & Test of Computers, vol. 19, no. 4, p. 131-132, 2002,.

[7] Kazmierczak, M. “White-Box Verification Techniques in a Networking ASIC Design”, Technical Report, Departament of Information Technology, Lund Institute of Technology, 2001.

[8] Axis Systems, “Assertion Processor”, August 2002.

[9] McMillan, K. L., “Symbolic Model Checking”, Kluwer Academic Publishers, 1993.

[10] Shimizu, K., Dill, D., Hu, A., “Monitor-Based Formal Specification of PCI”, Proc. Formal Methods in Computer Aided Design, pp 335-353, 2000.

[11] Switzer, S., Landoll, D., Anderson, T., “Functional Verification with Embedded Checkers”, 9th Annual International HDL Conference Proceedings, March 2000.

[12] IEEE Standard 1149.1-2001, ISBN 0-7381-2944-5.

[13] H. Foster, "Improving Verification through Property Specification", D&R Industry Articles.

[14] L. Bening, H. Foster, Principles of Verifiable RTL Design, Kluwer Academic Publishers, 2001.

[15] Foster, H. and Coelho, C., “Assertions Targeting a Diverse Set of Tools” International HDL Conference, 2001.

[16] Kantrowitz, M. and Noack, L. M., “I am Done Simulating: Now What? Verification Coverage Analysis and Correctness of the DECchip 21164 Alpha microprocessor,” 33rd. Design Automation Conference, 1996.

[17] Foster, H. et al, “Assertion-Based Design” , Kluwer Academic Publishers, to be released on 2003.