The Chip is Ready. Am I done? On-chip Verification using Assertion Processors

José Augusto M. Nacif
CS Dept. - UFMG - Brazil
jnacif@dcc.ufmg.br

Flávio M. de Paula
MindSpeed Technologies - USA
flavio.depaula@mindspeed.com

Claudionor N. Coelho Jr.
CS Dept. - UFMG - Brazil
coelho@dcc.ufmg.br

Fernando C. Sica
CS Dept.-UFMG-Brazil
sica@dcc.ufmg.br

Harry Foster
Jasper DA - USA
harry@jasper-da.com

Antônio O. Fernandes
CS Dept.-UFMG-Brazil
otavio@dcc.ufmg.br

Diógenes C. da Silva
EE Dept.-UFMG-Brazil
diogenes@cpdee.ufmg.br

Abstract

White-box verification is a technique that reduces observability problems by locating a failure during design simulation without the need to propagate the failure to the I/O pins. White-box verification in chip level designs can be implemented using assertion checkers to ensure the correct behavior of a design. With chip gate counts growing exponentially, today's verification techniques, such as white-box, can not always ensure a bug free design. This paper proposes an assertion processor to be used with synthesized assertion checkers in released products to enable intelligent debugging of deployed designs. Extending white-box verification techniques to deployed products helps locate errors that were not found during simulation / emulation phases. We present results of the insertion of assertion checkers and an assertion processor in an 8-Bit processor and a communication core.

1. Introduction

There has been a large number of design errors detected after the chip has been released [1–3]. As designs increase in complexity, it becomes clear that no one can ensure a bug-free design using conventional validation tools, including simulation / emulation and formal verification techniques. Probably the most famous is the Pentium Floating Point Bug [4] that was found just after chip deployment.

As design validation is moving beyond chip deployment, we propose in this paper a methodology to detect errors in released products based on the notion of assertion processor. An assertion processor is a circuit inserted into the design to monitor synthesized assertions, taking appropriate action in the case of an assertion failure.

This paper is outlined as follows. Section 2 describes the basis for this work, i.e. controllability and observability, and their relation to design validation. Section 3 presents assertion libraries that can be synthesized to facilitate on-chip debug. Section 4, describes the assertion processor framework. Finally, we present conclusion and future work.

2. Controllability and observability

The ability to test a design correlates to the ability of controlling and observing the behavior of a design. The increase of design complexity over the past years has weakened the ability to test a design, and even if a design error can be controlled, it may be very difficult to observe the error using the design I/O pins. This is a widely accepted problem in the integrated circuits industry and academic community, with numerous paper published in this subject [5–8].

The recent improvement of synthesis techniques allowed RTL-based designs from hardware description languages to be adopted by the designers as the design entry and methodology. Moreover, the use of RTL-based designs enabled more aggressive validation techniques based on White-box verification as opposed to Black-box verification.

Black-box verification relates 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.

Consider Figure 1 as an example. This figure presents an excerpt of a larger sequential design, with registers, represented by the two boxes and gates, represented by combinational logic cloud. By zooming in we find a combinational logic with two AND logic gates, Y and Z and one OR logic gate, X. In this figure xz and yz represents the internal interconnection of these logic gates. Table 1 presents the truth table for this excerpt.

By following the Black-box verification approach, in Figure 2, we would apply a test vector to the input and compare the output to the expected data. Consider now, that we have the input b stuck-at zero. We can easily see that by even testing all possible combinations of the inputs we would not be able to propagate the failing point to the output. Therefore, we would not observe the failure.

White-box verification is a technique used to validate a

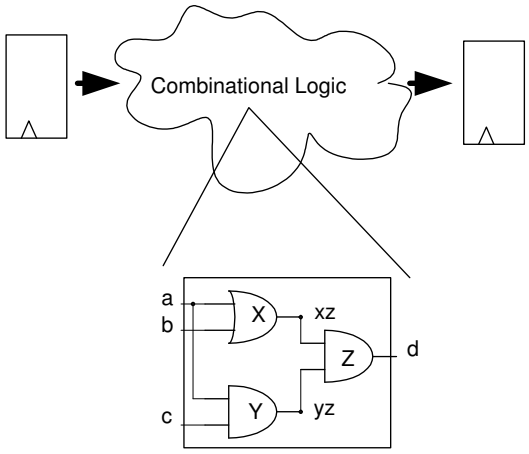

Figure 1. Sequential circuit and its combinational logic.

Table 1. Truth table of logic in Figure 1.

a	b	c	d
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

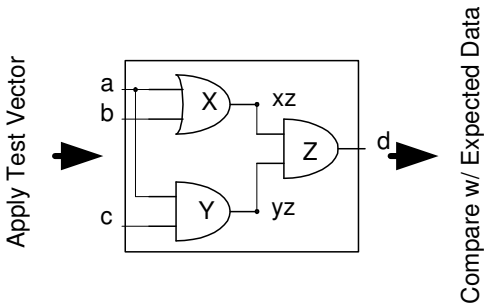

Figure 2. Black-Box verification approach.

design by inspecting internal wire connections of the design, thus improving the overall observability. When used with monitors, it provides a very powerful tool to aid design validation. An assertion monitor is a piece of HDL code that evaluates specific conditions on the designs' internal wires. 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 behaviors of internal design structures [9, 10]. 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 [9], from some pragmas of a specific tool such as in [10], or from a testbench written using a testbench language, such as OpenVera [11]. 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 [12, 13].

Consider applying the White-box verification approach in the example in Figure 3. First, we add an assertion to the wire connections of the design that we are concerned with the property we want to assert. For sake of this example, let us assume that the error condition occurs when $f1 + f2 > 1$. Although from the inspection of only this part of the circuit, we could clearly state that f1 and f2 can be true at the same time, making the assertion false, in general because of enviromental conditions $f1 + f2 > 1$ may never occur in a correct design.

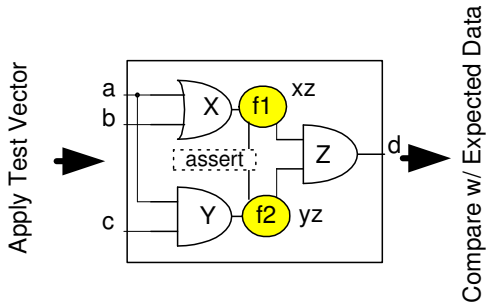

Figure 3. White-box verification approach.

White-box verification has been widely applied during the simulation, formal analysis and emulation [14–17] phases of a design. The initial goals of White-box verification are capturing the design intent and finding bugs as the design progresses. However, the use of White-box verification does not guarantee that a design is bug-free because of the complexity of a design. In the context of chip-level designs implemented in field-programmable gate arrays (FPGAs), assertion monitors enables reconfigurable designs to be monitored at run-time after deployment of the design. If a bug is ever found in the design, an assertion engine stores the error information that will later be notified to a designer. Because the error information is directly linked to an RTL design, the designer will be able to locate the problem faster, thus being able to provide a new version in a very short time. The previous work on synthesizing assertion checkers [18, 19] didn't address issues like wrong design assumptions or proposed an architecture that could inform which assertion had failed.

3. On-chip verification

An architecture for on-chip verification can be found in Figure 4. This architecture is based on three components: a sea of synthesizable assertions based on Open Verification Library (OVL) [20]; an assertion processor, which is a circuit designed to process the results of the assertions

and to take proper action, being as simple as a circuit that raises an error pin or as complex as an embedded processor that dispatches an error correction routine; and a routing mechanism that routes error information from the assertions to the assertion processor.

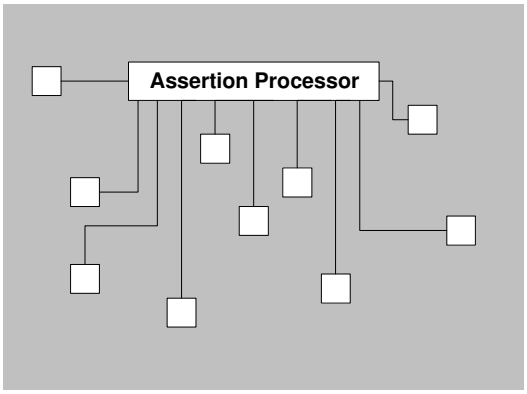

Figure 4. Diagram of assertion processor framework.

3.1. Synthesizable assertions with routing mechanisms

In [21], it was proposed an architecture for an assertion engine to be used in a reconfigurable design by extending the use of the White-box verification beyond the simulation/emulation phases of a design. The main idea was to modify the, OVL to support on-chip run-time debug. This modified library was obtained by adding a Boundary-scan [22] chain to the assertions. This library provided support to solve the assertion routing problems, although no assertion processor was used to provide the circuit with an intelligent mechanism to process the error condition.

Figure 5 (a) presents a typical assertion module from OVL. The modified version with scan-chain architecture is presented in 5 (b). Table 2 contains a description of each signal. We refer the reader to a throughout description of the modified library [21].

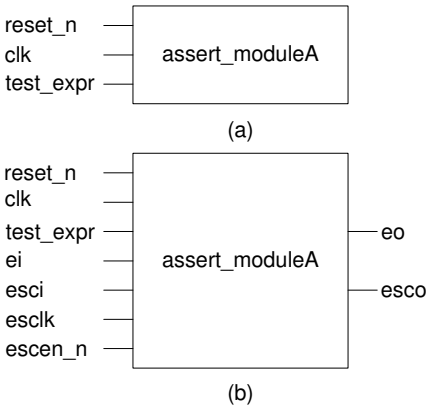

Figure 5. (a) Typical OVL assertion; (b) OVL assertion modified for scan-chain architecture.

Table 2. Signal descriptions for Figure 5(b).

Signal	Description	I/O
reset_n	Reset Active Low	Input
clk	System Clock	Input
test_expr	Any HDL test expression	Input
ei	Error Input	Input
esci	Error Scan Input	Input
eo	Error Output	Output
esco	Error Scan Output	Output
esclk	Error Clock	Input
escen_n	Error Scan Enable Active Low	Input

3.2. Generating chained assertions actual design

One of the major problems in modifying assertion instantiation into chained assertions stems from the fact that the circuit interface must be changed.

Figure 6 depicts an example of an 8051 ALU (Arithmetic Logic Unit) [23] hierarchical structure with chained assertions. White circles represent the original design hierarchy and dark ones the inserted assertions. Table 3 shows the assertion sequence that must be scanned from the pin *esco* in case an error is signaled by the *eo* pin. A typical timing diagram presenting the behavior of *eo*, *escen_n*, *esco*, and *esclk* pins are depicted in Figure 7.

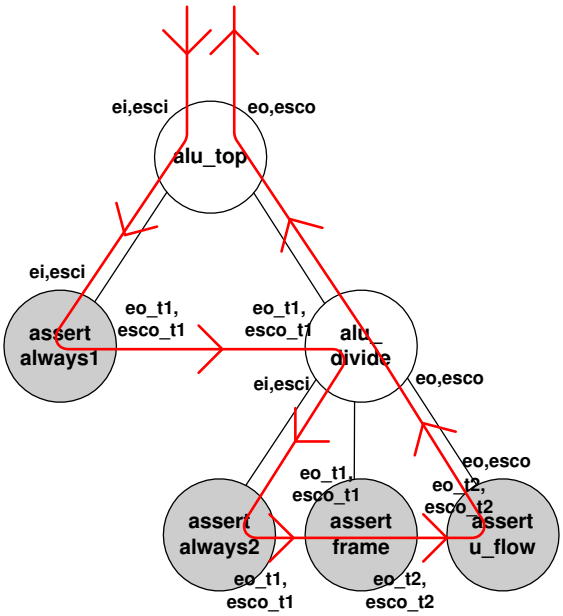

Figure 6. 8051's ALU hierarchy with assertion chaining.

Table 3. Assertion sequence list for Fig. 6.

Sequence Number	Assertion name
1	assert_uflow
2	assert_frame
3	assert_always2
4	assert_always1

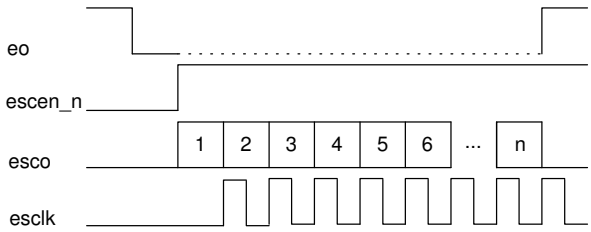

Figure 7. Behavior of the eo, escen_n, esco, and esclk pins.

4. Assertion processor architecture

Figure 8 present the proposed assertion processor with chained assertions. As it can be seen in the figure, an assertion processor minimally needs to perform three tasks:

- Scan the assertion chain to detect which assertion has caused the failure;
- Encode the possible tasks that must be performed for each assertion in the circuit;
- Perform specific tasks to overcome the error condition.

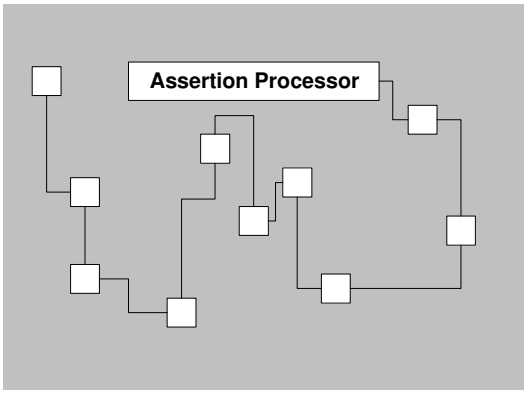

Figure 8. Assertion processor with chained structure.

Figure 9 presents a skeleton for a minimal Assertion Processor. This assertion processor contains three error processing tasks: halting the processor for more serious errors, resetting the entire chip, or performing a software interrupt to enable a processor core to perform a specific action. The reader should note that the priority for each assertion determining which action must be taken can be obtained directly from the OVL severity level.

Although this figure presents the minimum circuit for an assertion processor, more complex assertion processors can be implemented in the tasks execution part of the assertion processor. For example, if an assertion processor may interact with a network coprocessor if the error must be reported over an ethernet port.

```
module AssertionProcessor (...);
reg [‘LOGNOFASSERTIONS:0] count;
// SCAN DETECTION
always @(posedge clk) begin
  ...
  if (error detection)
  begin
    count = count + 1;
    if (esci == 1)
      ErrorNo = count;
    ...
  end
end
// PRIORITY ENCODING OF ERROR CONDITION
assign ErrorNo = ErrorEncoding(ErrorNo);
// ERROR CORRECTION
always @(posedge clk) begin
  if (error detected)
  begin
    casex (ErrorPriority)
    3’bxx1: // HALT INTEGRATED CIRCUIT
    3’bx1x: // HW RESET
    3’b1xx: // SW INTERRUPT
    endcase
  end
end endmodule
```

Figure 9. Assertion processor verilog HDL skeleton.

5. Results

This section presents the results of synthesizing assertion processors for an 8051 core [23] and for an I^2C (Inter Integrated Circuit) bus [24]. Although the proposed methodology focus in early design stages, the assertions were instantiated based on public domain specifications and the cores' documentation. The synthesis was performed using Xilinx Free Web Pack 5.2i environment. Xilinx Free Web Pack uses Xilinx Synthesis Technology (XST). Better results could be achieved using third party synthesis tools. The designs were synthesized and routed for a Virtex XCV300 FPGA using high area optimization effort.

Figure 10 shows the area in equivalent gate count for an assertion processor monitoring a number of assertions, supposing a priority encoding of five possible actions.

5.1. I^2C

I^2C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. I^2C standard was developed by Philips semiconductors [25]. Its applications include LCD drivers, remote I/O ports, RAM, EEPROM, data converters, digital tuning and signal processing circuits for radio and video systems, and DTMF generators.

In Table 4 some examples of assertions inserted in the I^2C core are shown. The total number of inserted asser-

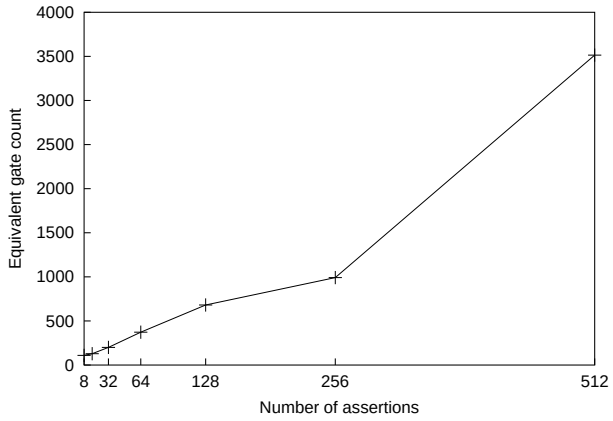

Figure 10. Assertion processor area increase supposing 5 priorities.

tions in the original design is 5. These assertions where inserted based on carefully reading and understanding of core's documentation.

Table 4. Assertions inserted in I^{2}C core.

Assertion type	Functionality
always	Ensures the correct interrupt request operation
never	Ensures that concurrent read and right signals will not occur
one_hot	Ensures the correct operation of control state machines

Table 5 presents the synthesis results for I^{2}C original design and using the proposed assertion processor architecture. Using total equivalence gate count as an example, we have an overhead of 37.27%. Considering that I^{2}C core complexity is relatively low, this results are acceptable. Although the circuit speed has dropped 100% from the original design speed, the normal operation of the design could still be maintained, as the chained structure is only active when *escen_n = 0*. As a result, we could run the time analyzer with the constrain *escen_n = 0*.

The reader should note also that the synthesizable OVL has been specified in RTL code. A handcrafted library, or a library with flip-flops with scan chain structures embedded could improve results considerably.

5.2. 8051 processor core

8051 is an 8-bit processor widely used in many embedded applications. There are several 8051 chip manufactories with different peripherals and memories configurations. The synthesized core has two 16-bit timer/counters, four 8-bit I/O ports, 4K bytes of on-chip program memory, and 128 bytes of on-chip data program (registers). Program memory and registers are inferred by synthesis tool using Virtex Flip-Flops. This memory structure consumes a significative part of design area. Table 6 show

Table 5. Synthesis results for I^{2}C communication core

Parameter	Original Structure	Chained Structure + AP	Overhead
Slice Flip Flops	122	129	5.74%
4 Input LUTs	221	357	61.54%
Slices	125	203	62.40%
Eq. gate count	2,557	3,510	37.27%
Maximum Frequency	89.17 MHz	45.17 MHz	97.41%

some assertions added to 8051 core. The total number of inserted assertions is 11.

Table 6. Assertions inserted in 8051 processor core.

Assertion type	Functionality
window	Verifies the division operation completion before a new enable signal
time	A four-clock-cycle ACK signal must be produced after an interrupt trigger
overflow	Ensures no stack overflow
always	Ensures that ALU always receives a valid opcode

Table 7 shows synthesis results for original 8051 processor core and using the proposed assertion processor architecture. Because of considerable complexity increase compared with I^{2}C the assertion overhead is significantly lower. Taking as a parameter the equivalent gate count, we have a 3.19% increase.

Table 7. Synthesis results for 8051 processor.

Parameter	Original Structure	Chained Structure	Overhead
Slice Flip Flops	838	943	12.53%
4 Input LUTs	4,487	4,698	4.70%
Slices	2,515	2,647	5.25%
Eq. gate count	68,141	70,313	3.19%
Maximum Frequency	12.99 MHz	12.86 MHz	1.01%

6. Conclusions and future work

White-box verification has been used as a better choice than Black-box verification for validating designs. This paper presented an assertion processor architecture to extend the use of White-box verification to the released

product. Synthesis results for two cores were presented. The use of assertion checkers added minimal overhead to presented designs. As future work, we are currently designing a self-reliant sensor network node using the architecture outlined in this paper.

7. Acknowledgments

This paper is supported under grants CNPq PNM #830107/2002-9, CNPq SensorNet #552111/2002-3, and CNPq/Pronex/SIAM #466079/2001-0.

References

[1] D. Beatty, *A Methodology for Formal Hardware Verification with Application to Microprocessors*. PhD thesis, Carnegie Mellon University, School of Computer Science, 1993.

[2] M. Kantrowitz and L. Noack, "I'm done simulating; now what? verification coverage analysis and correctness checking of the decchip 21164 alpha microprocessor," in *Proceedings of 33rd Design Automation Conference*, pp. 325–330, 1996.

[3] S. Taylor, M. Q. D. Brown, N. Dohm, N. Hildebrandt, J. Huggins, and C. Ramey, "Functional verification of a multiple-issue out-of-order, superscalar alpha processor - the dec alpha 21264 microprocessor," in *Proceedings of 35th Design Automation Conference*, pp. 638–643, 1998.

[4] M. Lowry and M. Subramaniam, "Abstraction for analytic verification of concurrent software systems," in *In Symp. on Abstraction, Reformulation, and Approx.*, 1998.

[5] S. Devadas and K. Keutzer, "A unified approach to the synthesis of fully testable sequential machines," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 4, pp. 39–51, 1991.

[6] H. Fujiwara, "Computational complexity of controllability/observability problems for combinational circuits," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 762–767, 1990.

[7] H. Fujiwara, *Logic Testing and Design for Testability*. The MIT Press, 1985.

[8] T. H. Chen and M. A. Breuer, "Automatic design for testability via testability measures," *IEEE Transctions on Computer-Aided Design*, vol. 4, no. 1, pp. 3–11, 1985.

[9] J. Bergeron, *Writing Testbenches Functional Verification of HDL Models*. Kluwer Academic Publishers, 2000.

[10] 0-In Design Automation, Inc., "Assertion-based verification for complex designs." The Verification Monitor, January 2002.

[11] Synopsys, Inc., "Assertion-based verification," May 2002.

[12] A. Gupta, "Assertion-based verification turns the corner," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 131–132, 2002.

[13] M. Kazmierczak, "White-box verification techniques in a networking asic design," tech. rep., Lund Institute of Technology, 2001.

[14] Axis Systems, "Assertion processor," August 2002.

[15] K. L. McMillan, *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[16] K. Shimizu, D. L. Dill, and A. J. Hu, "Monitor-based formal specification of PCI," in *Formal Methods in Computer-Aided Design*, pp. 335–353, 2000.

[17] S. Switzer, D. Landoll, and T. Anderson, "Functional verification with embedded checkers," in *9 th Annual International HDL Conference Proceedings*, 2000.

[18] M. Oliveira and A. Hu, "High-level specification and automatic generation of ip interface monitors," in *Proceedings of 39th Design Automation Conference*, pp. 129–134, 2002.

[19] R. Drechsler, "Synthesizing checkers for on-line verification of system-on-chip designs,," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. IV748–IV751, 2003.

[20] H. Foster and C. Coelho, "Assertions targeting a diverse set of tools," in *10 th Annual International HDL Conference Proceedings*, 2001.

[21] J. A. Nacif, F. M. de Paula, H. Foster, C. Coelho, F. C. Sica, D. C. da Silva, and A. O. Fernandes, "An assertion library for on-chip white-box verification at run-time," in *Proceedings of Latin American Test WorkShop*, 2003.

[22] IEEE, *Standard 1149.1-2001*. IEEE Press, 2001.

[23] S. Teran and J. Simsic, "8051 core." Available at http://www.opencores.org/projects/8051, November 2002.

[24] R. Herveille, "I^2C Controller Core." Available at http://www.opencores.org/projects/i2c, December 2002.

[25] Philips Semiconductors, "The I^2C-Bus Specification," 2000.