Table of Contents Search Author Index

Improved Assertion Lifetime via Assertion-Based Testing Methodology

Mohammad Riazati, Siamak Mohammadi,
Ali Afzali-Kusha, Zain Navabi
School of Electrical and
Computer Engineering
University of Tehran
Email: {m.riazati, smohammadi}@ece.ut.ac.ir,
afzali@ut.ac.ir, navabi@ece.neu.edu

***Abstract*—Assertions-based verification (ABV) has been widely used in digital design validation. Assertions are HDL-syntaxed representation of design specification and used as a functional error detection mechanism. During the process of designing with HDLs, assertions are imported which could fire in case of violation during testbench run. Although these assertions are mostly used during simulation and for verifying the functional correctness of the design, but as they illustrate the specifications of a design, it is likely that their lifetime could be extended by embedding them in the chip to detect low level faults like stuck-at faults. In this paper, we introduce a new automatable assertion-based on-line testing methodology. Experimental results show that the synthesis of assertions into a chip, and then using them for online testing, can provide an acceptable coverage for stuck-at faults.**

***Index Terms*— Assertion, OVL, on-line testing, fault coverage**

I. Introduction

Assertion-based verification is a variant of white box verification used in digital design and has been gaining much attention in literature and industry during the last several years [5]. The intended behavior of a design can be defined as a set of logical and timing relationship called properties. Properties are specification of assertions, static or temporal, that the implementation of DUV (Design Under Verification) has to honor. They express properties of the logic that hold true. Assertions can be utilized as an input to both simulation and static verification tools [3].

Different languages exist that can be used as assertion declaration. Property Specification Language (PSL), System Verilog Assertions (SVA), Open Verification Library (OVL) or even simple C++ and VHDL ASSERT statement can be used to define assertions in a design. These assertions formally encode the designer's assumptions or expectations of the design based on the functional intent [1]. Their domain is from designing a simple shift register or complicated queue to a microprocessor design. As the designer, must import these assertions during the design process, it is recommended to write carefully selected assertions that mostly cover critical functionalities of the specification because having less assertion causes faster simulation. In this paper we are going to use assertions to import Online Testing modules in the target IC. Although it is possible to use any kind of assertion for this purpose, we have chosen to use OVL assertions. They contain about 30 assertions, ranged from simple ones like *assert always* and *assert one hot*, to some more complicated ones like *assert cycle sequence*. OVL assertions can be used in both VHDL and Verilog. They are inserted as instantiations. Assertions can show the specification of a design. As an example to show the correspondence between high level design specification, and low level stuck-at fault detection, suppose a one hot style controller and an *assert_one_hot* assertion, embedded into the HDL to check its correctness. This assertion will check whether at the output there is only one '1' or not. After the simulation process is over, these assertions are normally removed and the synthesis process starts. But if we extend the lifetime of assertions by keeping them and trying to synthesize them as online testing modules, we can also detect possible stuck-at faults which may occur after manufacturing. Any stuck-at fault can change one of the output bits from '0' to '1' or vice versa. It is clear that this fault will possibly fire the assertion. Another advantage of this method is that it can even detect multiple-stuck-at faults.

In this paper, we introduce a method to show, how it is possible to make OVL assertions, synthesizable. As they are originally intended to be used in simulation process, they are not synthesizable. To use them in the synthesis process, it is necessary to remove some high level construct used in them like string errors and *$display* tasks. Another method would be to rewrite them at the gate level. Implementing them in gate level, gives us the ability to reduce the size of the synthesized modules.

In section II, we discuss some assertion related techniques for design evaluation and test. Section III introduces the proposed methodology. In section IV, some results are discussed and the summary and conclusion will be given in last section.

1-4244-0765-6/06/$20.00 ©2006 IEEE

II. RELATED WORKS

Today, the usual verification method for discovering design errors still is simulation [4]. There has been several researches on simulation based verification and use of assertions in the design phase. The purpose of the ABV [5] is to convert functional features of a specification into explicit hardware properties. Accellera, introduced OVL assertions to be used as embedded assertions in HDL (VHDL, Verilog) designs [6]. In [8] a new approach is introduced where the central focus of the verification process is on assertions, which detect bugs, guide the test writers to write better tests and direct testbenches to produce stimuli. In [10] a set of system level assertions have been defined. These assertions are automatically converted to monitor hardware or software during the system-level synthesis process depending on their type and also synthesis style of their corresponding functions. ACFC (Assertions for Control Flow Checking) are introduced in [12]. They use these assertions to detect functional faults.

Although there have been several researches on assertions and use of assertions in functional validation of designs, only a few of them have used assertions after the design has been simulated. [9] Presents a methodology to synthesize assertion monitors from visual specifications given in CESC (Clocked Event Sequence Chart). Boule´ and Zilic present an infrastructure for hardware emulation capable of supporting ABV [11]. They developed a tool that generates hardware assertion checkers for inclusion into efficient circuit emulation. Alex Orailoglu et al, discusses about design invariants, and concurrent testing via these invariants in [13]. Design invariants are some rules that must hold in the chip. The proposed method calculates the function g(x) and f(x) from primary inputs. The function g(x) expresses some invariance whereas f(x) is the output of the design. The correctness of the chip is tested by checking that the expected relation between g(x) and f(x) holds. Although this method uses only primary inputs of the design, (not the internal signals), it shows the possibility of having 75% to 100% fault coverage.

III. METHODOLOGY OVERVIEW

Fig. 1 shows a design with some assertions. Given a simple CPU, several OVL assertions modules (AMs) are embedded into the design. Each AM has inputs where some of them are design ports and others internal signals. AMs have typically no output port as they are normally represented in simulation through some strings.

Fig. 2 shows an overview of the proposed methodology where assertions are used as modules for on-line testing. The chip has an output signal that shows the firing of at least one of the assertions.

Some changes need to be done to the design as well as the OVL assertions, for them to become suitable for online testing. In the following paragraphs, we will describe these necessary changes in three steps.

1- All the signals being used as assertion module's inputs must be defined as module ports. One of the main advantages of assertions is that they are used as White Box testing. It means that they can use internal signals defined in design as their inputs. But in online testing method, the test module is preferably used as a separate module apart from the DUT module.

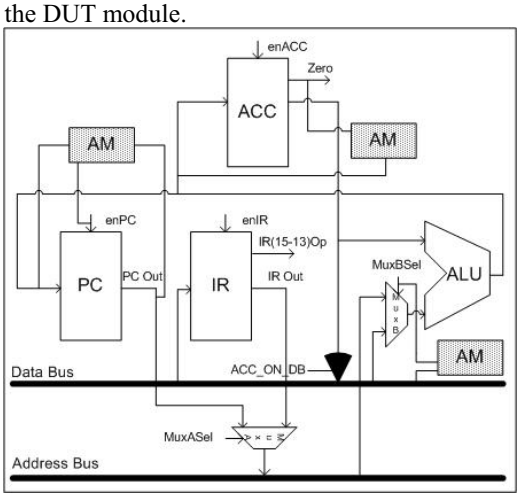

Fig. 1. Design with embedded assertions

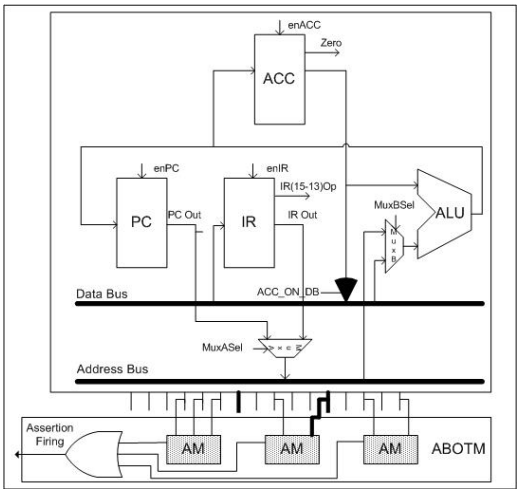

Fig. 2. Assertions used as online testing module

2- The design of assertion modules must change in order to have an output signal. Assertion modules do not have any output port. In presence of an incorrect design behavior when an assertion fires, the error string is written, via some system functions, to the output in simulation environment. For our purpose these functions and strings must be removed before the synthesis process (the next step), and the firing of assertion must be asserted by a signal to the outside environment of the design. This output gets the value '1' whenever the assertion is going to fire. This will be equivalent to the times when an error message was written to the standard output during the simulation. Having multiple assertions modules in ABOTM (Assertion Based Online Testing Module), the output of ABOTM will be the OR of these signals.

Fig. 3 shows the synthesizable 8-bit version of *assert_one_hot*:

3- Assertions were initially intended to be used during simulation. Therefore the coverage has not been so

important. It has been possible to add several assertions to the design to cover almost all functional errors. On the other hand, synthesizing several assertion modules could have an unexpected area overhead. Using high fault coverage assertions for on-line testing, reduction of number of assertions with respect to fixed area of each assertion, and distributing the assertions correctly throughout the design [13] will lead to lower area overhead and better fault coverage.

We define a metric (EAP) to evaluate an embedded assertion in the design which we will call Embedded Assertion Performance:

*EAP = (1 – Area Overhead) * Fault Coverage*

Area overhead is found after synthesizing the assertion modules and DUT module.

```
module assert_one_hot_synthesis(assert, clk, reset_n,
test_expr);
  output assert;
  reg assert;
  input clk, reset_n;
  input [7:0] test_expr;

  wire [7:0] test_expr_1 =
    test_expr - {{7{1'b0}},1'b1};
  always @(posedge clk)
  begin
    if ((test_expr == {8{1'b0}}) ||
      (test_expr & test_expr_1) != {8{1'b0}})
    begin
      assert = 1;
    end
    else
    begin
      assert = 0;
    end
  end
endmodule
```

Fig. 3. Synthesizable 8 bit assert_one_hot

Table I shows the calculation with the above formula for a sample design which has four embedded assertions. Fault coverage is obtained using a fault simulation tool. In our experiment, we used the fault simulator introduced in [2]. Navabi et al, proposed this fault simulator based on concurrent fault simulation implemented in VHDL. The fault simulation is performed by VHDL gate models capable of propagating faults in fault queues. Gate models process the fault queues and propagate them in delta time units. In these models, gates with faulty input values are expanded in delta time to evaluate faulty output values and propagate them to other sections of the circuit.

According to Table I for example, the lowest cost assertion to be inserted would be the assertion #3.

TABLE I
DECISION MAKING TO INSERT ASSERTION MODULES

Assertion Id	Area Overhead	Stuck-at fault coverage	EAP
#1	12%	15%	13.20%
#2	35%	47%	30.55%
#3	24%	53%	40.28%
#4	47%	40%	21.20%

Using multiple assertions, the total coverage may be less than the sum of fault coverage of each one except in the case where assertions would cover separate parts of the design. As a result, it is necessary to run fault simulation, for all combination of assertions. For example, we have inserted both assertions #1 and #3 and then performed the fault simulation. The area overhead of having both assertions adds up to 36% as we had expected, whereas the fault coverage resulted in only 57% which was less than the sum of their respective coverage (68%).

IV. EXPERIMENTAL RESULTS

We tested this approach on some designs from [7], with minor changes in some of them, to evaluate our proposed methodology. The fault coverage and area overhead are imposed by ABOTM. There are samples in [7] which introduce the capabilities of OVL assertions. Each design contains one or two assertions. We have synthesized and embedded these assertions into the designs.

Table II shows the results of synthesis, fault simulation and eventually EAP calculation.

Designs “Walking_One” and “Gray_Counter” both use the assertion *Assert_one_hot.* We have accidentally obtained the same stuck-at faults coverage for both of them. Note that despite the same coverage results, they have different performance due to their different area overhead.

Also note on Table II for the design “Sequence Machine” the two first assertions *Assert_cycle_seq1* and *Assert_next.* The second assertion, as we can see, exhibits a lower coverage but higher performance. This is also due to their different area overhead.

TABLE II
ON-LINE TESTING APPROACH APPLIED TO DESIGNS

Design	Assertion	Stuck-at fault coverage	EAP
BCD_Counter	Assert_always	33.21%	41.73%
Walking_One	Assert_change	37.75%	27.52%
	Assert_one_hot	***87.5%***	67.35%
Gray_counter	Assert_one_hot	***87.5%***	58.29%
Seq_Machine	Assert_cycle_seq1	53.27%	39.14%
	Assert_next	***33.46%***	***41.73%***
	Assert_cycle_seq2	46.75%	41.46%

This shows how important it could be to write fewer but best fit assertions in order to reach a higher performance on-line test.

V. CONCLUSION

Assertion-based verification is becoming a method widely used in digital design. The ability to test the functionality of a circuit is an increasingly important property of modern designs. We have introduced an automatable methodology where synthesizable assertions are embedded into the HDL design and used for on-line testing to detect stuck-at faults. We have shown that by writing judicious assertions, we are able to enhance the process of inserting on-line testing

modules into chip. A good assertion would cover more aspects of the design with low overhead. This methodology was applied to a number of designs and results were shown.

REFERENCES

[1] J. Bergeron. “Writing Testbenches: Functional Verification of HDL Models”, Kluwer Academic Publishers, New York, NY.
[2] M. Zolfy, S. Mirkhani, Z. Navabi, “Adaptation of an event-driven simulation environment to sequentially propagated concurrent fault simulation,” in Design, Automation and Test in Europe, 2001. Proceedings, March 2001 pp. 823
[3] A. Ziv, “Cross-product functional coverage measurement with temporal properties-based assertions [logic verification],” in Design, Automation and Test in Europe Conference, 2003, pp. 834-839.
[4] J. Cortéz1, D. Torres, “Design And Verification Based On Assertions: Some Statistics,” in 2nd International Conference on Electrical and Electronics Engineering (ICEEE) and XI Conference on Electrical Engineering (CIE), 2005
[5] H. Foster, A. Krolnik, D. Lacey, Assertion-Based Design, Kluwer Academic Publishers, Second Edition, 2004.
[6] http://www.accellera.org/
[7] Z. Navabi, Verilog Digital System Design: RT Level Design, Test and Verification. McGraw Hill Company, N.Y., New York, 2005.
[8] Synopsys, "Assertion-Based Verification", March 2003
[9] A. A. Gadkari, S. Ramesh, “Automated Synthesis of Assertion Monitors using Visual Specifications,” in Design, Automation and Test in Europe, 2005. Proceeding, pp. 390-395
[10] S. Hessabi, A. M. Gharehbaghi, B. H. Yaran, M. Goudarzi, “Integrating assertion-based verification into system-level synthesis methodology,” in The 16th International Conference on Microelectronics, ICM 2004 Proceedings., pp. 232-235
[11] M. Boule, Z. Zilic, “Incorporating Efficient Assertion Checkers into Hardware Emulation,” in the International Conference on Computer Design, 2005, pp. 221-228
[12] Rajesh Venkatasubramanian; J. P. Hayes, B. T. Murray, “Low-Cost On-Line Fault Detection Using Control Flow Assertions,” in the On-Line Testing Symposium, IOLTS, 2003, pp. 137-143
[13] Y. Makris, I. Bayraktaroglu, A. Orailoglu, “Enhancing Reliability of RTL Controller-Datapath Circuits via Invariant-Based Concurrent Test,” in the IEEE Transactions on Reliability, Volume 53, Issue 2, June 2004 pp. 269 - 278