ON-CHIP PROPERTY VERIFICATION USING ASSERTION PROCESSORS

José Augusto M. Nacif,[1] Claudionor Nunes Coelho Jr.,[1] Harry Foster,[2] Flávio Miana de Paula,[3] Edjard Mota,[4] Márcia Roberta Falcão Mota,[5] and Antônio Otávio Fernandes[1]

[1] *Computer Science Department*
Universidade Federal de Minas Gerais, MG, Brazil
{jnacif,coelho,otavio}@dcc.ufmg.br

[2] *Jasper Design Automation*
Mountain View, CA, USA
harry@jasper-da.com

[3] *MindSpeed Technologies*
Newport Beach, CA, USA
flavio.depaula@mindspeed.com

[4] *INDT - Instituto Nokia de Tecnologia, AM, Brazil*
edjard.mota@indt.org.br

[5] *Computer Science Department*
Universidade Federal do Amazonas, AM, Brazil
marcia.roberta@gmail.com

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.

Keywords: On-line verification, assertion processor, assertion-based verification

Please use the following format when citing this chapter:
Nacif, José Augusto, M., Coelho, Claudionor Nunes, Jr., Foster, Harry, Flávio, Miana de Paula, Mota, Edjard, Roberta Falcão Mota, Márcia, Fernandes, Antônio, O., 2006, in IFIP International Federation for Information Processing, Volume 200, VLSI-SOC: From Systems to Chips, eds. Glesner, M., Reis, R., Indrusiak, L., Mooney, V., Eveking, H., (Boston: Springer), pp. 101-117.

1. Introduction

There has been a large number of reported design errors detected after the chip has been released, such as in Beatty, 1993; Kantrowitz and Noack, 1996; Taylor et al., 1998. As design complexity increases, it becomes clear that no one can ensure a bug-free design using conventional validation tools using simulation, emulation and formal verification techniques. Probably the most famous bug reported in the literature is the Pentium Floating Point Bug Lowry and Subramaniam, 1998 that was found just after chip deployment.

As design validation clearly becomes one of the most critical issues in chip design today, we propose in this chapter a methodology that pushes design validation beyond chip deployment by using the notion of an 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 based on PSL and OVL 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. 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 Devadas and Keutzer, 1991; Fujiwara, 1990; Fujiwara, 1985; Chen and Breuer, 1985.

Recent improvements of synthesis techniques allowed RTL-based designs to be adopted as the main design capture methodology used by designers. Moreover, the use of RTL-based designs enabled more aggressive validation techniques based on White-box verification as opposed to Black-box verification Foster et al., 2004.

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. In addition to this problem, the failure may only be noticed several thousands of cycles after it has actually happened, making it difficult to detect and even to recognize the failure.

Consider Figure 1 as an example. This Figure presents an excerpt from a larger sequential design, with registers, represented by the two boxes and gates,

represented by combinational logic cloud. In this design, one of the operation modes of the design reduces a portion of the combinational logic to the circuit outlined in the picture, with two AND logic gates, Y and Z and one OR logic gate, X. In this figure, xz and yz represents the internal interconnection wires of these logic gates.

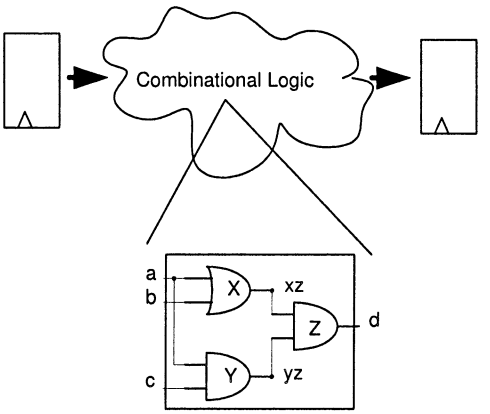

Figure 1. Sequential circuit and its combinational logic.

In this example, let us consider that we want to use the Black-box methodology to test a stuck-at-zero condition at pin b in this piece of logic. First, we apply a set of test vectors to the inputs and compare the output d to the expected value. Even though we can exercise all possible input vectors to this piece of logic, it is not be observable because the logic is redundant on this operation mode.

White-box verification is a technique used to validate a 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 Bergeron, 2000; 0-In Design Automation, Inc., 2002. 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 Bergeron, 2000, from some pragmas of a specific tool such as in 0-In Design Automation, Inc., 2002, or from a testbench written using a testbench language, such as Open-

Vera Synopsys, Inc., 2002. 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 Gupta, 2002; Kazmierczak, 2001.

Consider applying the White-box verification approach in the example in Figure 2. First, we add an assertion correlating the expected behavior of the wires. For the 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.

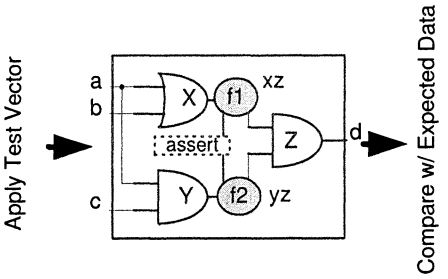

Figure 2. White-box verification approach.

White-box verification has been widely applied during the simulation, formal analysis and emulation Axis Systems, 2002; McMillan, 1993; Shimizu et al., 2000; Switzer et al., 2000 phases of a design. The initial goals of White-box verification are to capture the design intent, to document interface assumptions and to find bugs as the design progresses. However, the use of White-box verification does not guarantee that a design is bug-free because of the overall complexity of the 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 can be later 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 Oliveira and Hu, 2002; Drechsler, 2003 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 3. This architecture is based on three components: a sea of synthesizable assertions based on Open Verification Library (OVL) Foster and Coelho, 2001 or synthesized from PSL Accellera, 2004; 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.

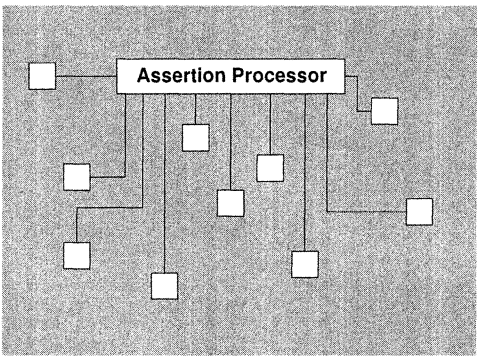

Figure 3. Diagram of assertion processor framework.

3.1 Synthesizable assertions with routing mechanisms from OVL

In Nacif et al., 2003, 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 IEEE, 2001 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 4 (a) presents a typical assertion module from OVL. The modified version with scan-chain architecture is presented in 4 (b). Table 2 contains a description of each signal. We refer the reader to a throughout description of the modified library Nacif et al., 2003.

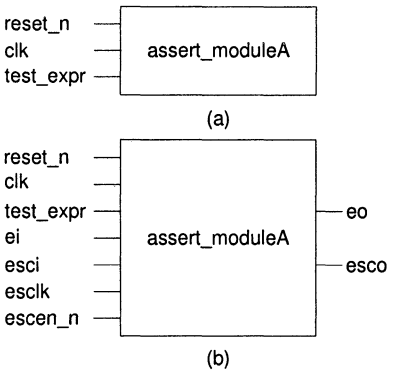

Figure 4. (a) Typical OVL assertion; (b) OVL assertion modified for scan-chain architecture.

Table 1. Signal descriptions for Figure 4(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 Synthesizable assertions from PSL

The Accellera Property Specification Language (PSL) Accellera, 2004 is an ideal language for specifying complex design intent in either linear-time temporal logic or in branching-time temporal logic.

The PSL language definition is segmented into the following layers: Boolean, temporal, modelling, and verification. The temporal layer supports either the linear-time temporal logic or branching-time temporal logic operators. In this section, we consider only a linear-time temporal logic component. For a more complete definition, see Accellera, 2004.

At the Boolean layer, a PSL specification references signals and variables within an HDL description (for example, Verilog or VHDL). Hence, the un-

derlying HDL syntax and semantics for Boolean expressions ensure semantic consistency between the property specification and the HDL model.

Sequences of Boolean conditions that occur at successive clock cycles can be described succinctly using Sequential Extended Regular Expressions (SEREs). Sequences and SEREs can be constructed as follows (where *b* is a Boolean expression):

- `b` : a Boolean expression is a SERE in its simplest form,
- `{SERE}` : a sequence constructed by a SERE,
- `SERE ; SERE` : a SERE constructed by concatenating two SEREs,
- `{sequence | sequence}` : a sequence describing alternative sequences,
- `{sequence & sequence}` : a sequence describing parallel non-length matching sequences (that is, two sequences, both hold at the current cycle, regardless of whether they complete in the same cycle or in different cycles),
- `{sequence && sequence}` : a sequence describing parallel length matching sequences (that is, two sequences, both hold at the current cycle, and both complete in the same cycle).

PSL provides various repetition operators ([]) that concisely describe repeated concatenation of the same SERE.

For example, given the SERE `r` and a Boolean `b`:

- `r[*m:n]` : a sequence of `m` to `n` contiguous occurrences of `r`,
- `b[=m:n]` : any sequence containing from `m` to `n` occurrences of `b`,
- `b[->m:n]` : any sequence ending in the mth to nth occurrence of `b`.

The repeat range `m:n` can be replaced by a single constant `n` (for example, `[*2]`). In addition, an unbounded range could be expressed as `[*0:inf]`, where the keyword `inf` represents infinity.

PSL supports all the standard LTL operations. In addition, more readable operators are defined in terms of the base operators. For example, given the PSL temporal formulas `f`, `f1`, `f2`:

- `!f` : `f` does not hold,
- `f1 & f2` : `f1` and `f2` both hold,
- `f1 | f2` : `f1` or `f2` or both hold,
- `f1 -> f2` : `f1` implies `f2`,

- `f1 <-> f2 : f1 -> f2 and f2 -> f1`,
- `always f` : `f` holds in every cycle,
- `never f` : `f` does not hold in any cycle,
- `next f` : `f` holds in the next cycle,

PSL also supports operators to build complex properties out of SEREs, such as `{r1} |-> {r2}`, meaning that `{r2}` starts in the last cycle of `{r1}`, and `{r1} |=> {r2}`, meaning that `{r2}` starts in the first cycle after `{r1}`, as wells as a way to define named properties, which facilitates reuse.

A property ensuring that `a` and `b` are mutually exclusive can be specified as:

```
property mutex = always !(a <-> b) @(posedge clk);
```

When we synthesize assertions from PSL, we consider only a synthesizable subset of the PSL language, i.e. without any behavior that leads to infinite memory. The synthesis process generates an RTL description of the design that instantiates an OVL assertion. For example, the PSL assertion described below leads to the circuit of Figure 5.

```
assert always {e1; e2; e3} |-> e4 @(posedge clk)
```

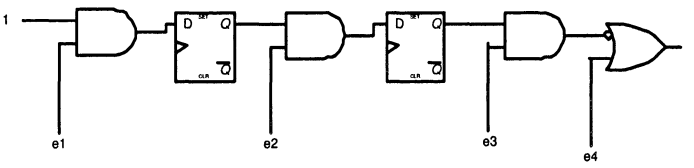

Figure 5. Synthesized circuit from PSL

This circuit corresponds to the following Verilog-HDL description:

```
reg a1, a0;
always @(posedge clk) a0 <= e1 & 1;
always @(posedge clk) a1 <= e2 & a0;
assert_always (reset, clk, e4 | ~(e3 & a1));
```

3.3 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) Teran and Simsic, 2002 hierarchical structure with chained assertions. White circles represent the original design hierarchy and dark ones the inserted assertions. Table 2 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 is depicted in Figure 7.

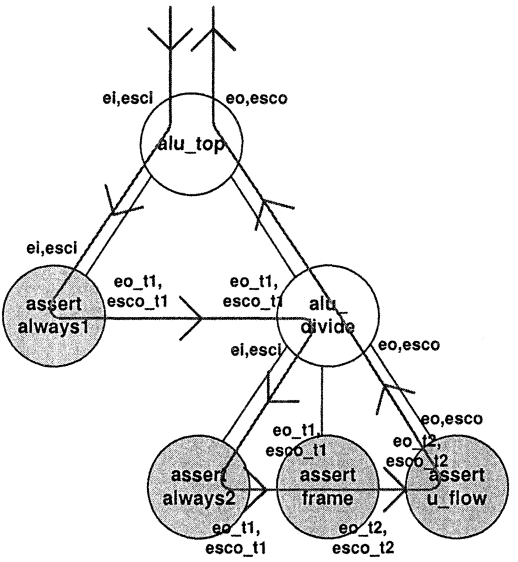

Figure 6. 8051's ALU hierarchy with assertion chaining.

Table 2. Assertion sequence list for Fig. 6.

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

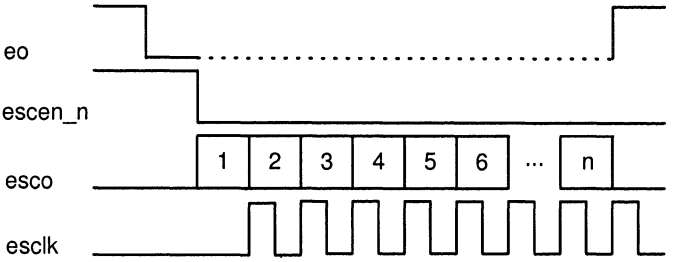

Figure 7. Behavior of the eo, escen_n, esco, and esclk pins.

4. Assertion processor architecture

Figure 8 presents 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.

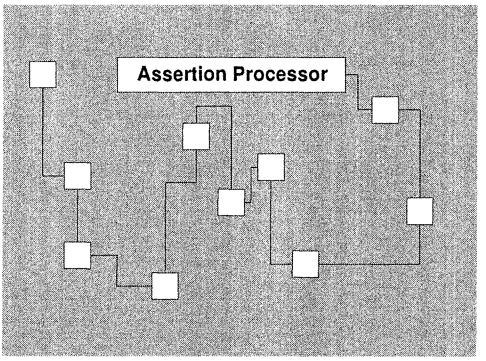

Figure 8. Assertion processor with chained structure.

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.

```
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.

Although this Figure presents the minimum circuit for an assertion processor, more complex assertion processors can be implemented in the tasks exe-

cution 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.

In order to automatize the assertion chaining process XRoach tool has been developed Oliveira et al., 2003. XRoach processes verilog hierarchical designs with OVL and PSL assertions. It compiles the design and links it to the modified synthesis version of OVL. XRoach output files are the verilog design with chained assertions and a list of the assertions in the chained order, that is used by the assertion processor to identify which assertion had failed. Figure 10 presents XRoach user interface. The basic assertion synthesis flow is shown below:

1 Design with PSL and OVL assertions

2 Convertion of PSL properties into RTL code + OVL assertions

3 Selection of assertions to be synthesized

4 Selection of severity level for each synthesized assertion

5 Synthesis of scanning structure for instantiated assertions

6 Synthesis of assertion processor skeleton

7 Generation of new design hierarchy with assertion scan-in/out + assertion processor

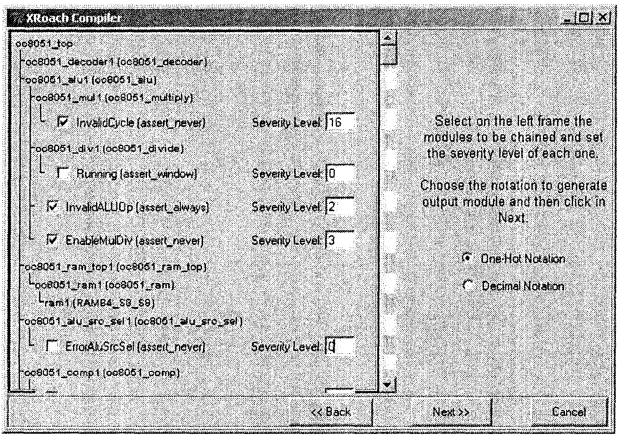

Figure 10. XRoach user interface.

5. Results

This section presents the results of synthesizing assertion processors for an 8051 core Teran and Simsic, 2002 and for an I^{2}C (Inter Integrated Circuit) bus Herveille, 2002. 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 11 shows the area in equivalent gate count for an assertion processor monitoring a number of assertions, supposing a priority encoding of five possible actions.

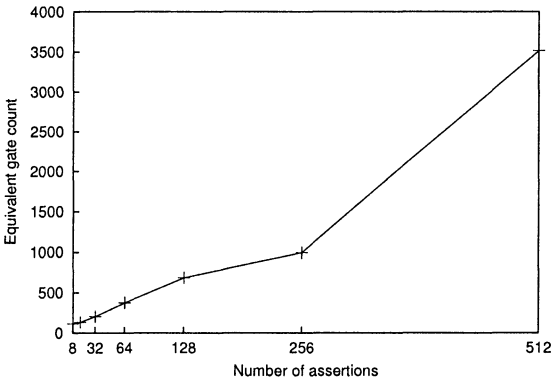

Figure 11. Assertion processor area increase supposing 5 priorities.

5.1 I^{2}C

I^{2}C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. I^{2}C standard was developed by Philips semiconductors Philips Semiconductors, 2000. 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 3 some examples of assertions inserted in the I^{2}C core are shown. The total number of inserted assertions in the original design is 5. These as-

sertions where inserted based on carefully reading and understanding of core's documentation.

Table 3. Assertions inserted in I^2C core.

Assertion type	Functionality
assert_always	Ensures the correct interrupt request operation
assert_never	Ensures that concurrent read and write signals will not occur
assert_one_hot	Ensures the correct operation of control state machines

Table 4 presents the synthesis results for I^2C 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^2C 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.

Table 4. Synthesis results for I^2C 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%
Equivalent gate count	2,557	3,510	37.27%
Maximum Frequency	89.17 MHz	45.17 MHz	97.41%

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 sig-

nificative part of design area. Table 5 shows some assertions added to 8051 core. The total number of inserted assertions is 11.

Table 5. Assertions inserted in 8051 processor core.

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

Table 6 shows synthesis results for original 8051 processor core and using the proposed assertion processor architecture. Because of considerable complexity increase compared with I^2C the assertion overhead is significantly lower. Taking as a parameter the equivalent gate count, we have a 3.19% increase.

Table 6. Synthesis results for 8051 processor core.

Parameter	Original Structure	Chained Structure + AP	Overhead
Slice Flip Flops	838	943	12.53%
4 Input LUTs	4,487	4,698	4.70%
Slices	2,515	2,647	5.25%
Equivalent gate count	68,141	70,131	3.19%
Maximum Frequency	12.99 MHz	12.86 MHz	1.01%

6. Conclusions and future work

We presented a technique that enhances currently validation capability of assertion and property based techniques beyond deployment of chip-level designs. This technique can be applied to inumerous situations, including emulation phase of designs, in which validation testcases execute at full speed and for fault-tolerant chip design, in which an assertion failure can yield to mission failure.

We showed that a decision circuit that monitors assertions in a design can be used to validade the design. This circuit, which was called an assertion processor, upon the detection of an assertion failure can dispatch a set of recovery procedures, ranging from hardware reset, software reset or event put the chip into halt mode.

Along with the assertion processor, we generated a version of the Open Verification Library (OVL) fully synthesizable, and a procedure to generate digital circuits from PSL descriptions by converting them into RTL code along with an OVL assertion.

Since the total number of assertions can be very high, we presented a tool called XRoach which enabled designers to select which assertions he/she wanted to synthesize into the final circuit. These assertions were concatenated to the assertion processor, and a skeleton of the assertion processor was automatically generated.

An example of the tool usage was shown using an 8051 processor, showing that minimum overhead in circuit size was obtained by carefully selecting proper assertions. As future works, we intend to investigate different concatenation procedures between assertions and the assertion processors, and to work in the interaction between software and hardware failures.

Acknowledgments

We thank Leonardo Otaviano, Márcia Oliveira and Fernando Sica for their help in the preparation of an earlier version of this paper. We also thank CNPq for financial support under grants PNM #830107/2002-9 and SensorNet #552111/2002-3.

References

0-In Design Automation, Inc. (2002). Assertion-based verification for complex designs. The Verification Monitor.

Accellera (2004). Proposed Standard Property Specification Language (PSL) 1.1.

Axis Systems (2002). Assertion processor.

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

Bergeron, J. (2000). *Writing Testbenches Functional Verification of HDL Models*. Kluwer Academic Publishers.

Chen, T. H. and Breuer, M. A. (1985). Automatic design for testability via testability measures. *IEEE Transctions on Computer-Aided Design*, 4(1):3–11.

Devadas, Srinivas and Keutzer, Kurt (1991). A unified approach to the synthesis of fully testable sequential machines. *IEEE Transactions on Computer-Aided Design*, 10(4):39–51.

Drechsler, R. (2003). Synthesizing checkers for on-line verification of system-on-chip designs,. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages IV748–IV751.

Foster, Harry and Coelho, Claudionor (2001). Assertions targeting a diverse set of tools. In *10 th Annual International HDL Conference Proceedings*.

Foster, Harry, Krolnik, Adam C., and Lacey, David J. (2004). *Assertion-Based Design*. Kluwer Academic Publishers.

Fujiwara, H. (1985). *Logic Testing and Design for Testability*. The MIT Press.

Fujiwara, H. (1990). Computational complexity of controllability/observability problems for combinational circuits. *IEEE Transactions on Computers*, 39(6):762–767.

Gupta, Aarti (2002). Assertion-based verification turns the corner. *IEEE Design & Test of Computers*, 19(4):131–132.

Herveille, Richard (2002). I^2C Controller Core. Available at: http://www.opencores.org/projects/i2c.

IEEE (2001). *Standard 1149.1-2001*. IEEE Press.

Kantrowitz, M. and Noack, L. (1996). 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*, pages 325–330.

Kazmierczak, Marcin (2001). White-box verification techniques in a networking asic design. Technical report, Lund Institute of Technology.

Lowry, M. and Subramaniam, M. (1998). Abstraction for analytic verification of concurrent software systems. In *In Symp. on Abstraction, Reformulation, and Approx.*

McMillan, Kenneth L. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers.

Nacif, José Augusto, de Paula, Flávio Miana, Foster, Harry, Coelho, Claudionor, Sica, Fernando Cortez, da Silva, Diógenes Cec'ilio, and Fernandes, Antônio Otávio (2003). An assertion library for on-chip white-box verification at run-time. In *Proceedings of Latin American Test WorkShop*.

Oliveira, M. and Hu, A. (2002). High-level specification and automatic generation of ip interface monitors. In *Proceedings of 39th Design Automation Conference*, pages 129–134.

Oliveira, Márcia M., Nacif, José Augusto, Coelho, Claudionor N., and Fernandes, Antônio Otávio (2003). XRoach: A Tool for Generation of Embedded Assertions. In *Proceedings of Chip in Sampa Student Forum*.

Philips Semiconductors (2000). The I^2C-Bus Specification.

Shimizu, Kanna, Dill, David L., and Hu, Alan J. (2000). Monitor-based formal specification of PCI. In *Formal Methods in Computer-Aided Design*, pages 335–353.

Switzer, S., Landoll, D., and Anderson, T. (2000). Functional verification with embedded checkers. In *9 th Annual International HDL Conference Proceedings*.

Synopsys, Inc. (2002). Assertion-based verification.

Taylor, S., Brown, M. Quinn D., Dohm, N., Hildebrandt, N., Huggins, J., and Ramey, C. (1998). Functional verification of a multiple-issue out-of-order, superscalar alpha processor - the dec alpha 21264 microprocessor. In *Proceedings of 35th Design Automation Conference*, pages 638–643.

Teran, Simon and Simsic, Jaka (2002). 8051 core. Available at: http://www.opencores.org/projects/8051.