‘concurrent assertion’

it’s a vacuous world !

Concurrent Assertion – *without* an implication

```
sequence sr1;
  req ##2 gnt;
endsequence

property pr1;
  @(posedge clk) sr1;
endproperty

reqGnt: assert property (pr1)   $display($stime,,,"\t\t %m PASS");  else
                                $display($stime,,,"\t\t %m FAIL");
```

Simulation Log

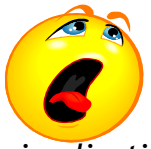
```
# run -all
#         10  clk=1 req=0 gnt=0
#         10            test_basic_property.reqGnt FAIL
#         30  clk=1 req=1 gnt=0
#         30            test_basic_property.reqGnt FAIL
#         50  clk=1 req=1 gnt=0
#         70  clk=1 req=0 gnt=0
#         70            test_basic_property.reqGnt FAIL
#         90  clk=1 req=0 gnt=1
#         90            test_basic_property.reqGnt FAIL
#         90            test_basic_property.reqGnt PASS
#        110  clk=1 req=0 gnt=0
#        110            test_basic_property.reqGnt FAIL
```

Whenever 'req' is Low, the assertion FAILs !!

That's because, a sequence simply says that 'req' be true at the clock edge and that gnt must be true 2 clocks later.

It does *NOT* say check the sequence "Only *If* 'req' is true at posedge clk".

But you really don't care for result when 'req' is Low.



That's where an implication operator comes into picture…

Concurrent Assertion – *with* an implication

```
sequence sr1;
  ##2 gnt;
endsequence

property pr1;
          @(posedge clk) req  |->  sr1;
endproperty

reqGnt: assert property (pr1)   $display($stime,,,"\t\t %m PASS");  else
                                $display($stime,,,"\t\t %m FAIL");
```

IMPLICATION OPERATOR (OVERLAPPING)

ANTECEDENT

CONSEQUENT

Simulation Log

```
#        10  clk=1   req=0 gnt=0
#        10          test_basic_property1.reqGnt PASS
#        30  clk=1   req=1 gnt=0
#        50  clk=1   req=0 gnt=0
#        50          test_basic_property1.reqGnt PASS
#        70  clk=1   req=0 gnt=1
#        70          test_basic_property1.reqGnt PASS
#        70          test_basic_property1.reqGnt PASS
#        90  clk=1   req=1 gnt=0
#       110  clk=1   req=0 gnt=0
#       110          test_basic_property1.reqGnt PASS
#       130  clk=1   req=0 gnt=0
#       130          test_basic_property1.reqGnt FAIL
```

In this example, we moved a part of the sequence to the property and are using it as an antecedent to imply a consequent.

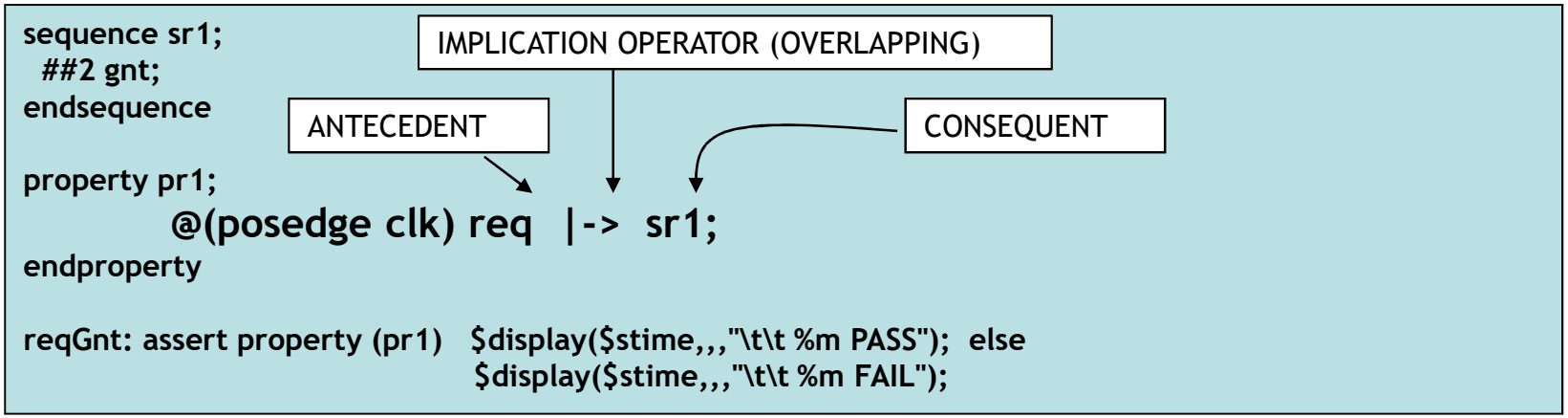
With an implication operator, the 'antecedent' MUST be TRUE to evaluate the consequent. Hence, whenever 'req' is Low, the antecedent is false and the implication simply does not fire and there is no failure message as in the previous example.

BUT WAIT…

Now the property PASSes whenever 'req' is Low.

What's going on???

Concurrent Assertion – *Vacuous Pass – What ??*



LRM 3.1a (Page 232) ::

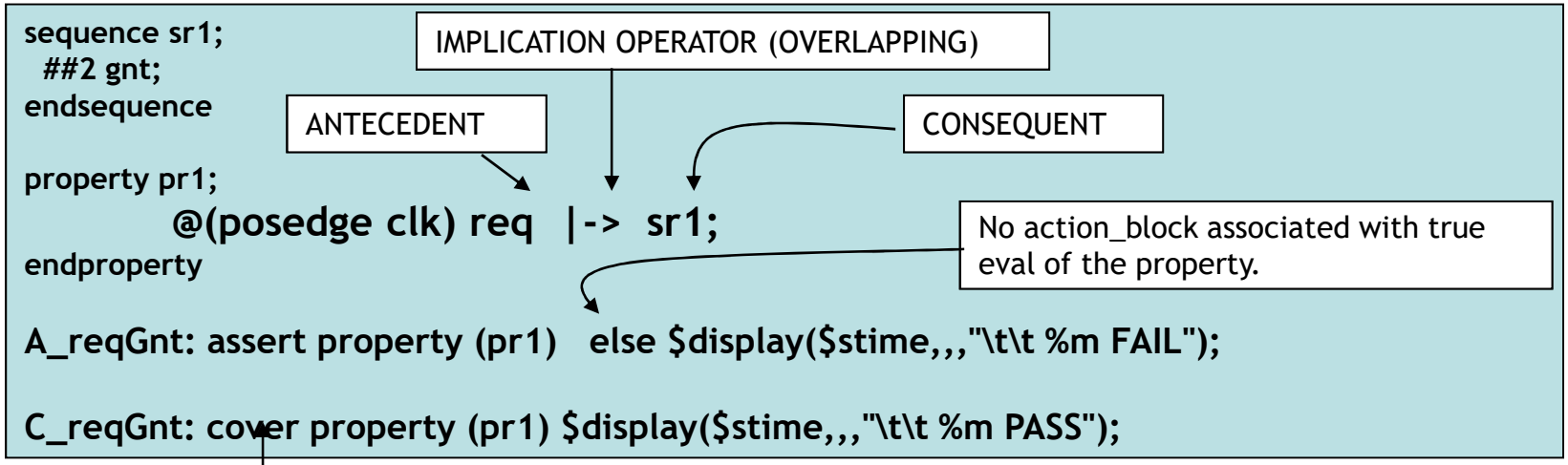
"If there is no match of the antecedent sequence_expr, then evaluation of the implication succeeds vacuously and returns true"

A couple of ways to get around this…

One is to simply not use the action_block associated with 'pass' (duh…) of the property, so that you don't get pass indication vacuously

But what if you do want to know when the property passes…

Concurrent Assertion – with 'cover'… take 1…



You may use a 'cover' statement to cover the same property that is asserted. 'cover' does not report vacuous pass ***(but note that many simulators require a run time option to 'filter' out vacuous pass on a 'cover')***.

Note that 'cover' does not allow an action_block if the property fails.

```
# run -all
#        10  clk=1   req=0 gnt=0
#        30  clk=1   req=1 gnt=0
#        50  clk=1   req=0 gnt=0
#        70  clk=1   req=0 gnt=1
#        70            test_basic_property2.C_reqGnt PASS
#        90  clk=1   req=1 gnt=0
#       110  clk=1   req=0 gnt=0
#       130  clk=1   req=0 gnt=0
#       130            test_basic_property2.A_reqGnt FAIL
```

In this example, we removed the action block associated with the true (i.e. pass) evaluation of the property to avoid the vacuous $display.

If you do need an action block for a match (i.e. Pass) of a property, you may use a 'cover' statement to cover the same property that is asserted.

Concurrent Assertion – with 'cover'… take 2…

OK, what if your simulator does not filter for a vacuous pass on a 'cover'.

There are many ways to get around it. Here's one. 'cover' the sequence/property -without- an implication.

```
property pr1;
            @(posedge clk) req  |->  ##2 gnt;
endproperty

property pr2;
            @(posedge clk) req ##2 gnt;
endproperty

A_reqGnt: assert property (pr1)  else $display($stime,,,"\t\t %m FAIL");
C_reqGnt: cover property (pr2) $display($stime,,,"\t\t %m PASS");
```

No action_block associated with true eval of the property.

```
# run -all
#         10  clk=1   req=0 gnt=0
#         30  clk=1   req=1 gnt=0
#         50  clk=1   req=0 gnt=0
#         70  clk=1   req=0 gnt=1
#         70            test_basic_property2.C_reqGnt PASS
#         90  clk=1   req=1 gnt=0
#        110  clk=1   req=0 gnt=0
#        130  clk=1   req=0 gnt=0
#        130            test_basic_property2.A_reqGnt FAIL
```

For 'cover' we removed the implication operator. Vacuous pass applies only when there is an implication operator and the antecedent does not match.

Without an implication operator there is no vacuos pass and since there is no 'failure' action_block with a 'cover' you get a pass indication only when the property/sequence matches…

CUSTOMER TESTIMONIALS …

"Ashok is a very good instructor – very impressive."

John Reykjalin, President, Grizzly Peak Engineering, Inc.

"The class was excellent, well distributed between the fundamentals and the practical examples. With a little tweak, we can use those example assertions presented right now in our design development! In addition I would like to thank you for seminar associated text material. The handout (book) is a few levels above anything I have previously seen. The rules and example descriptions cover the associated topic completely."

Thomas Slee, Sr. Electrical Engineer, ASIC Verification Lead, Space System Loral

"The seminar on SVA was very educative and informative. The material was in-depth, was from a hardware design/verification person's perspective and it was vendor neutral. The information was very good and I hope to have a chance to use it in the future."

Shubha Umesh, Senior Logic Engineer, LeCroy Corporation

"I like the seminar very much since it's packed full of technical explanations and examples of System Verilog Assertion. You slides are also easy to follow and understand."

Gloria Chen, ASIC Verification Engineer, 3PAR

"Your seminar clearly showed us that System Verilog Assertions provide new powerful interfaces and how they are implemented as part of the language. Your seminar was one of the best seminar I ever had."

Aki Niimura-san, Ponderosa Design

Ashok B. Mehta

- **30+ years of experience in SoC, CPU design and verification at DEC, Data General, Intel, Applied Micro, TSMC**

- **Author of two books**
  - SystemVerilog Assertions and Functional Coverage (2nd edition)
    - A comprehensive guide to languages, methodology and applications
  - ASIC/SoC Functional Design Verification
    - A comprehensive guide to technologies and methodologies

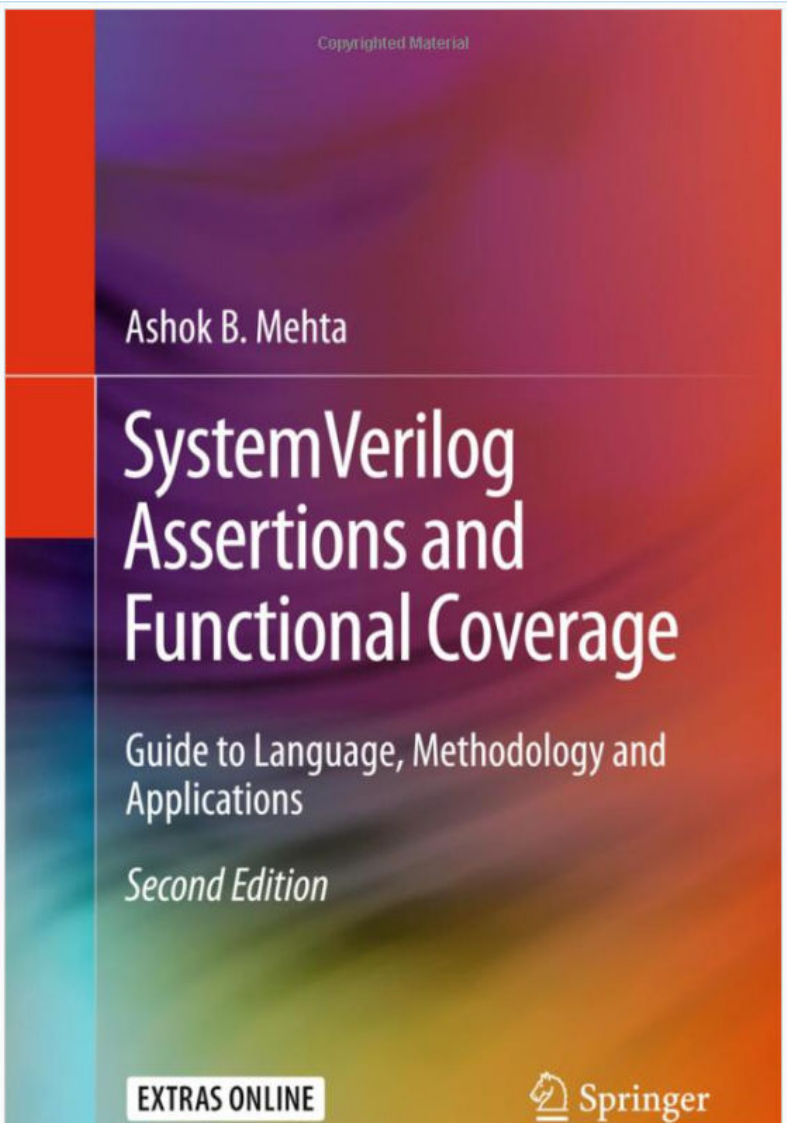
- **17 US Patents on 3DIC and SoC verification**

- **Expertise in:**
  - Coverage Driven Verification (CDV),
  - Assertion Based Verification (ABV),
  - Universal Verification Methodology (UVM),
  - Constrained Random Verification,
  - Behavioral/Architectural modeling,
  - Static Formal verification,
  - Hardware Acceleration,
  - ESL/Virtual Platform (TLM 2.0), etc.

Visit www.defineview.com

Ashok B. Mehta – 17 issued US patents

	PAT. NO.		Title
1	9,646,128	T	System and method for validating stacked dies by comparing connections
2	9,625,971	T	System and method of adaptive voltage frequency scaling
3	9,612,277	T	System and method for functional verification of multi-die 3D ICs
4	9,552,448	T	Method and apparatus for electronic system model generation
5	9,514,268	T	Interposer defect coverage metric and method to maximize the same
6	9,404,971	T	Circuit and method for monolithic stacked integrated circuit testing
7	9,158,881	T	Interposer defect coverage metric and method to maximize the same
8	9,110,136	T	Circuit and method for monolithic stacked integrated circuit testing
9	9,047,432	T	System and method for validating stacked dies by comparing connections
10	9,015,649	T	Method and apparatus for electronic system model generation
11	8,972,918	T	System and method for functional verification of multi-die 3D ICs
12	8,966,419	T	System and method for testing stacked dies
13	8,826,202	T	Reducing design verification time while maximizing system functional coverage
14	8,578,309	T	Format conversion from value change dump (VCD) to universal verification methodology (UVM)
15	8,522,177	T	Method and apparatus for electronic system function verification at two levels
16	8,402,404	T	Stacked die interconnect validation
17	8,336,009	T	Method and apparatus for electronic system function verification at two levels



This book provides a hands-on, application-oriented guide to the language and methodology of both SystemVerilog Assertions and SystemVerilog Functional Coverage.

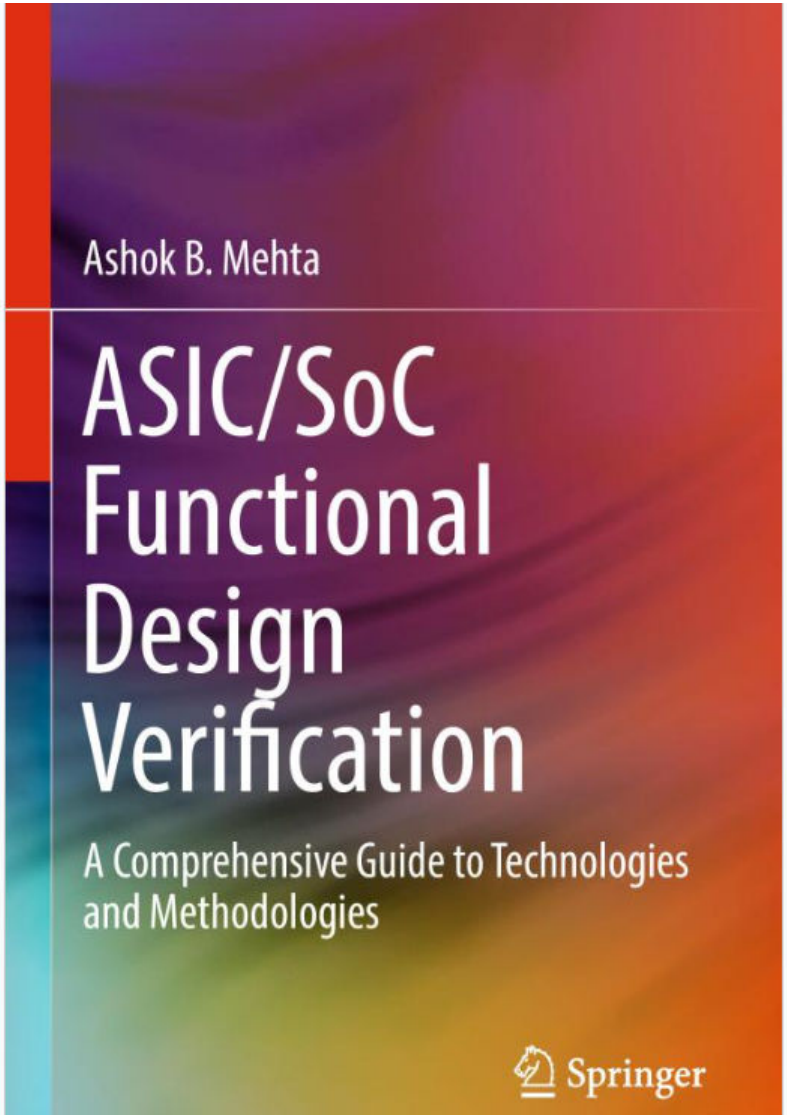
Readers will benefit from step-by-step approach to functional hardware verification using SystemVerilog Assertions and Functional Coverage.

The book has a strong end-user perspective, which makes it plenty easy to digest complex features and apply them with ease to a design.

Plenty of real-life applications

This is an excellent Reference Book

The book will enable design verification engineers to uncover hidden and hard to find bugs, point directly to the source of the bug, provide for a clean and easy way to model complex timing checks and objectively answer the question ‘have we functionally verified everything’.



This book describes in detail all required technologies and methodologies needed to create a comprehensive, functional design verification strategy and environment.

The author describes industry standard technologies such as

UVM (Universal Verification Methodology)

SVA (SystemVerilog Assertions)

SFC (SystemVerilog Functional Coverage)

CDV (Coverage Driven Verification)

Low Power Verification (Unified Power Format UPF)

AMS (Analog Mixed Signal) verification

Virtual Platform TLM2.0/ESL (Electronic System Level) methodology

Static Formal Verification

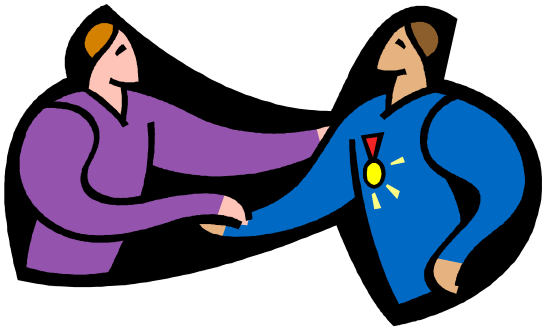
LEC (Logic Equivalency Check)

Hardware Acceleration

Hardware Emulation, Hardware/Software Co-verification

PPA (Power Performance Area) analysis on a virtual platform

Reuse Methodology from Algorithm/ESL to RTL, and other overall methodologies



happy asserting…

For a detailed training that takes you through System Verilog Assertions and Functional Coverage language and methodology step by step with examples/simulation logs, real life applications and practical LABs please contact…

DefineView Consulting
www.defineview.com

ashok_mehta@yahoo.com