
SystemVerilog Assertions for Mixed-Language Support

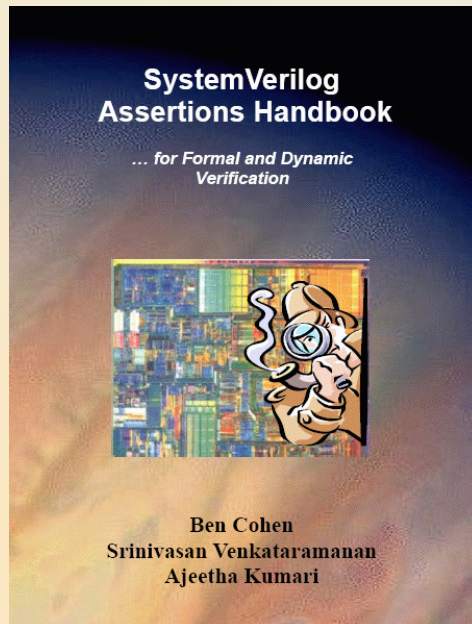
Ben Cohen
VhdlCohen Publishing
Trainer / Consultant / Author



Ben Cohen <http://www.abv-sva.org>

ben@abv-sva.org

SystemVerilog Assertions Handbook Now Available!



- Ben Cohen Trainer, Consultant, Publisher (310) 721-4830**
- 1. SystemVerilog Assertions Handbook, 2005 isbn 0-9705394-7-9**
 - 2. Using PSL/SUGAR for Formal and Dynamic Verification 2nd Edition, 2004, isbn 0-9705394-6-0**
 - 3. Real Chip Design and Verification Using Verilog and VHDL, 2002 isbn 0-9705394-2-8**
 - 4. Component Design by Example, 2001 isbn 0-9705394-0-1**
 - 5. VHDL Coding Styles and Methodologies, 2nd Edition, 1999 isbn 0-7923-8474-1**
 - 6. VHDL Answers to Frequently Asked Questions, 2nd Edition, 1998 isbn 0-7923-8115**



Joined Sutherland HDL <http://www.sutherland-hdl.com/>
for Training in:
SystemVerilog Assertions and
SystemVerilog for Verification



Ben Cohen <http://www.abv-sva.org>

ben@abv-sva.org

Outline

- **Why SVA**
 - For Verilog / SystemVerilog
 - for VHDL
- **How VHDL fits in SV / SVA Environment**
 - External property module with bind
 - Inline SVA with VHDL
- **Adoption Issues**
- **Conclusions and Recommendations for Interoperability**



Why SVA for Verilog /SystemVerilog

- **SVA are part of SystemVerilog !**
 - Interoperable with the design (Verilog / mixed HDL) & TB
- **Specifies properties and sequences**
 - May use local variables
 - Properties can be recursive
 - in context of SystemVerilog
- **Provides rich set of Operators**
 - Property operators
 - Sequence operators
- **Supports assertions**
 - Concurrent
 - Procedural Assertions
 - One-shot
 - Non-blocking concurrent in Procedural Block
 - Blocking with expect
 - Immediate

SVA Specific

SVA Specific



Why SVA for Verilog /SystemVerilog

- **Provides Assertion-Based System Functions**
 - Sampled Valued Functions (Value access, value change)
 - Vector-Analysis System Functions
 - Severity-level System Functions
 - Assertion-Control System Tasks
- **Provides method calls from within a thread**
 - User-defined functional coverage
 - User-defined messages
 - Reactive testbench
- **Supports Single and Multi-Clocking Sequences**

SVA Specific



Why SVA for Verilog /SystemVerilog

- **Can create reactive testbenches with Action blocks**

SVA Specific

- Can activate a task upon assertion success or failure
 - Can use SVA API to dynamically query assertion coverage/status
- **Can be declared in various SystemVerilog units**
 - **Can bind to Verilog / VHDL modules**
 - **Can be used for dynamic and formal verification**



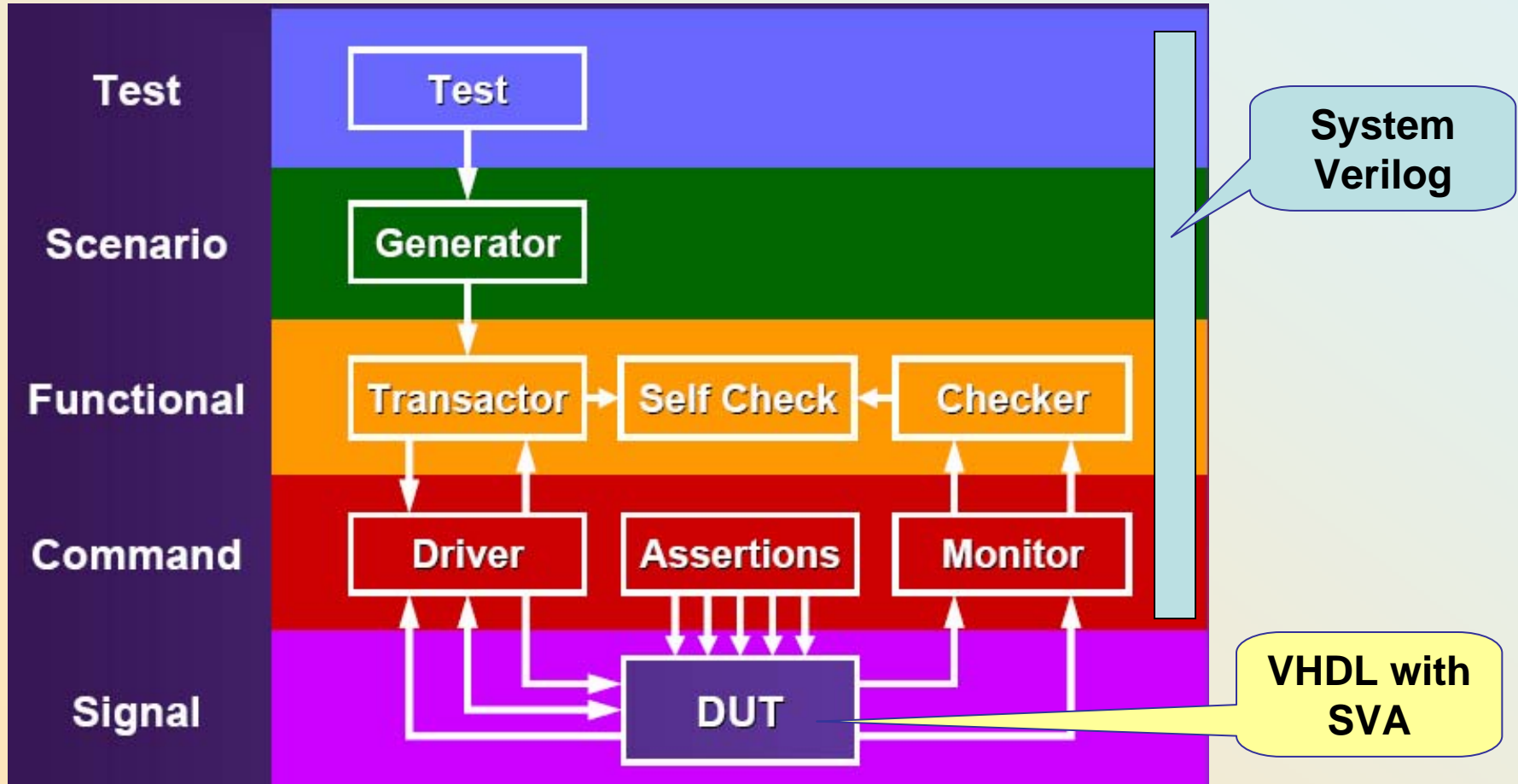
Why SVA for VHDL Designs

- **Many users expressed a preference to SV / SVA**
 - Need to add ABV to current VHDL projects
 - Looking toward a smooth migration to SV / SVA for future projects
 - Need to build verification environment in SystemVerilog
 - Like SystemVerilog capabilities
 - Like the features and reactivity of SVA
 - Want to adopt RVM
(*Reference Verification Methodology*)
 - Can be used for dynamic and formal verification



How VHDL Fits in SV / SVA Environment

RVM Verification Environment Layers

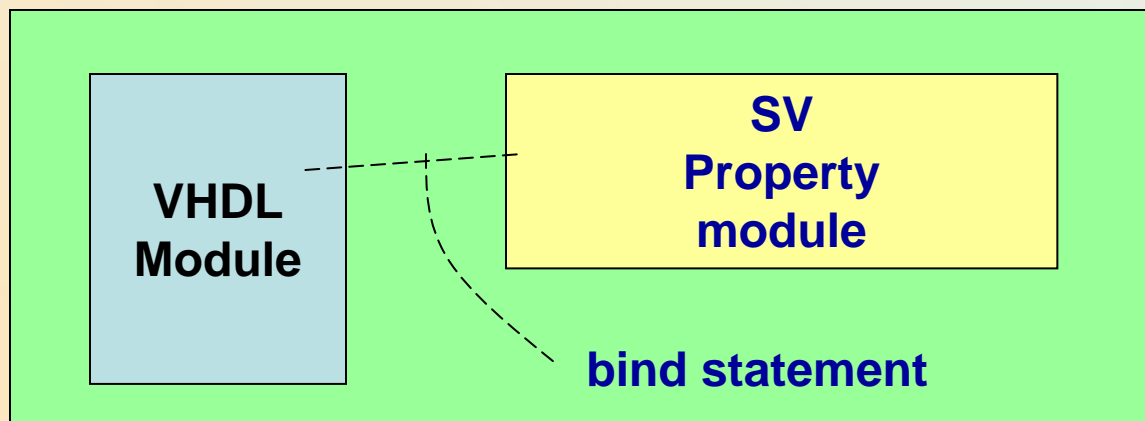


How VHDL Fits in SV / SVA Environment

- **Two solutions today with no modification to SystemVerilog**

- **External property module with bind**

- Implemented today by Synopsys
- Similar to SV bind methodology to SV module



- **Inline SVA with VHDL**



Example: VHDL Module with No Assertions

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity counter is
port (
  count_out : out std_logic_vector(7 downto 0);
  ld_enb    : in  std_logic;
  ...
  count_enb : in  std_logic);
end entity counter;
```

Port signals

```
architecture rtl of counter is
  signal count : unsigned(7 downto 0); -- counter reg
  signal tc   : std_logic; -- terminal count
  ...
begin -- architecture rtl
  ...
end architecture rtl;
```

Internal signals



SystemVerilog Property Module Assertions for the VHDL Module

```
module counter_props (  
  input logic[8-1:0] count_out, // counter output  
  input ld_enb,  
  ... // reset, active hi  
  input [7:0] count, // INTERNAL signal  
  input tc // INTERNAL signal  
);  
timeunit 1ns; timeprecision 100ps;  
parameter MAXCOUNT = 8'hFF;
```

```
property p_tc;  
  @ (posedge clk) disable iff (!rst_n)  
    count==0 && $past(count)==MAXCOUNT && !ld_enb | => tc;  
endproperty : p_tc  
ap_tc : assert property(p_tc) else  
  $display ("0t error in terminal count", $time);
```

```
endmodule : counter_props
```



Bind Statement in testbench

```
bind counter counter_props counter_props_1(  
    .count_out(count),  
    .data_in(data_in),  
    .ld_enb(ld_enb),  
    .count_enb(count_enb),  
    .clk(clk),  
    .rst_n(rst_n),  
    .count(count),  
    .tc(tc));
```



Why Inline SVA with VHDL

- **Inline ABV is part of methodology**
 - **Documentation**
 - Requirements
 - Assumptions
 - **Code reviews**
 - **Verification**



Proposed VHDL Inline Solution

Note: Tool may hide implementation details

VHDL with inline sva pragmas

```
architecture rtl of Y is
...
begin
-- sva assert property
(@ (posedge clk) req => [0:4] ack);
..
end architecture rtl;
```

Creation
of bind
module
& bind

Testbench

bind
statement

Bind
module



VHDL with Inline SVA Example

Pragmas Identify SVA

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity counter is
port (
    count_out : out std_logic_vector(7 downto 0); -- counter output
    ...
    count_enb : in std_logic);      -- count enable, active hig
end entity counter;
```

```
architecture rtl of counter is
    signal count : unsigned(7 downto 0); -- counter reg
    signal tc : std_logic; -- terminal count
```

```
...
begin -- architecture rtl
    -- sva property p_tc;
    -- @ (posedge clk) disable iff (!rst_n)
    -- count==0 && $past(count)==MAXCOUNT && !ld_enb |=> tc;
    -- endproperty : p_tc
    -- sva ap_tc : assert property(p_tc) else $display ("0t error in terminal count", $time);
    ...
end architecture rtl;
```



Generated SystemVerilog Bind Module

```
module counter_props (  
  input logic[8-1:0] count_out, // counter output  
  input ld_enb,  
  ... // reset, active hi  
  input [7:0] count, // INTERNAL signal  
  input tc // INTERNAL signal  
);  
timeunit 1ns; timeprecision 100ps;  
parameter MAXCOUNT = 8'hFF;  
  
property p_tc;  
  @ (posedge clk) disable iff (!rst_n)  
    count==0 && $past(count)==MAXCOUNT && !ld_enb |=> tc;  
endproperty : p_tc  
ap_tc : assert property(p_tc) else  
  $display ("0t error in terminal count", $time);  
endmodule : counter_props
```



Generated Bind Statement

```
bind counter counter_props counter_props_1(  
    .count_out(count),  
    .data_in(data_in),  
    .ld_enb(ld_enb),  
    .count_enb(count_enb),  
    .clk(clk),  
    .rst_n(rst_n),  
    .count(count),  
    .tc(tc));
```



Adoption Issues for Inline SVA

- **Support of data types**
 - Enumeration, records, multi-dimensional arrays, generics
- **Integration of packages**
 - Type definitions
 - Use of functions
- **SVA reactivity to environment**
 - No reactive assignment to VHDL objects
 - Possible assignment to SV variable
 - display of messages and value of signals
 - Severity level in action block
- **Only concurrent assertions allowed**
 - Procedural and immediate assertions disallowed



Conclusions and Recommendations for Interoperability

- **External SV property module with bind to VHDL component is feasible today**
- **inline SVA with VHDL with pragma mechanism is straightforward**
 - Is the fast route to provide VHDL with additional features (for design and verification) while preserving the language standard.
 - PSL uses the same pragma mechanism.
- **Recommend adoption of these methods**
 - In the realm of SystemVerilog
 - Interoperability among vendors
 - Consistent with bind of SV modules to other modules



Questions ?



Ben Cohen <http://www.abv-sva.org>

ben@abv-sva.org