Binding SystemVerilog to VHDL Components Using Questa

Allan Crone
Mentor Graphics Corporation

Mentor Graphics®

SystemVerilog offers a rich set of testbench automation capabilities, native assertions, and functional coverage. These features make SystemVerilog increasingly appealing to VHDL users who need to implement an efficient and effective functional verification methodology. The Questa™ advanced verification environment offers native support of SystemVerilog, Verilog, and VHDL, making it easy to adopt SystemVerilog.

The flexibility provided by Questa's single kernel architecture enables VHDL users to easily attach SystemVerilog to existing VHDL components. The connection is made using the SystemVerilog *bind* function, which gives modules and program blocks access to both external ports and internal signals in the bound VHDL object. Among other things, you can use this capability to monitor VHDL functionality with SystemVerilog Assertions (SVA) or assess the functional coverage of VHDL designs.

Monitoring VHDL Components

Suppose you want to use SVA to monitor a VHDL component. To do this effectively, you need SVA to access the component's ports and internal signals *without* modifying the VHDL code.

For example, let's take the following VHDL code:

```
entity cpu is
port (
          a : in std_logic;
          b : in std_logic;
…
architecture rtl of cpu is
…
signal c : std_logic;
begin
…
end rtl;
```

First you define the properties to be monitored using a SystemVerilog program:

```
program cpu_props(input d,e,f);
…
assert property (@(posedge d) e |-> ##[1:2] f );
…
endprogram
```

Next you tie the SVA to the VHDL component by creating a simple wrapper module:

```
module sva_wrapper;

  bind cpu                      // Bind to all occurrences of cpu

  cpu_props cpu_sva_bind        // Bind cpu_props to cpu and call this
                                // instantiation cpu_sva_bind

  (.d(a),.e(b),.f(c));          // Connect the SystemVerilog ports to
                                // VHDL ports (a and b) and to the
                                // internal signals (c)
endmodule
```

To enable the actual binding in the simulator, you have two options. You can instantiate the *sva_wrapper* as follows:

```
…
— the dut
i_cpu : cpu port map (…=>…);

— the SystemVerilog assertions via the bind wrapper
i_sv : sva_wrapper;
…
```

Or you can simply load multiple top modules into the simulator:

```
vlib work
vlog *.*v
vcom *.vhd
vsim top sva_wrapper
```

You should keep in mind the following important points:

- Due to different module resolving rules in the Verilog and VHDL LRMs, all modules must be resolved prior to being referenced. Therefore, in the example above, you must compile all Verilog modules before compiling the VHDL.
- Because the VHDL LRM specifies that elaboration is performed in a lexical order, in VHDL you must instantiate the SystemVerilog wrapper module after the VHDL component is instanced.
- You can bind to all objects defined in the bound object.

Binding to Enumerated Types

SystemVerilog infers an enumeration concept similar to VHDL enumerated types. In VHDL, the enumerated names are assigned to a fixed enumerated value, starting left-most with the value 0. In SystemVerilog, you can also explicitly define the enumerated values.

Suppose you want to use SVA to monitor a VHDL finite state machine that uses enumerated types. With Questa, you can map VHDL enumerated types to Verilog integer or real types. This enables binding to VHDL enumerated types.

Consider the following VHDL code:

```
…
  type fsm_state is(idle, send_bypass,
                    load0,send0, load1,send1, load2,send2,
                    load3,send3, load4,send4, load5,send5,
                    load6,send6, load7,send7, load8,send8,
                    load9,send9, load10,send10,
                    load_bypass, wait_idle);
  signal int_state : fsm_state;
  signal nxt_state : fsm_state;
…
```

First you define the properties to be monitored in the SystemVerilog module. Then you map the vector to the enumerated name. Because *fsm_state* has 26 values, you need a 5-bit vector for an input port:

```
typedef enum {idle, send_bypass,
              load0,send0, load1,send1, load2,send2,
              load3,send3, load4,send4, load5,send5,
              load6,send6, load7,send7, load8,send8,
              load9,send9, load10,send10,
              load_bypass, wait_idle} fsm_state;

module interleaver_props (
  input clk, in_hs, out_hs,
  input [4:0] int_state_vec
);

fsm_state int_state;

assign int_state = int_state_vec; // Map vector to enum name
…
// Check for sync byte at the start of a every packet
property pkt_start_check;
  @(posedge clk) (int_state == idle && in_hs) -> (sync_in_valid);
endproperty
…
```

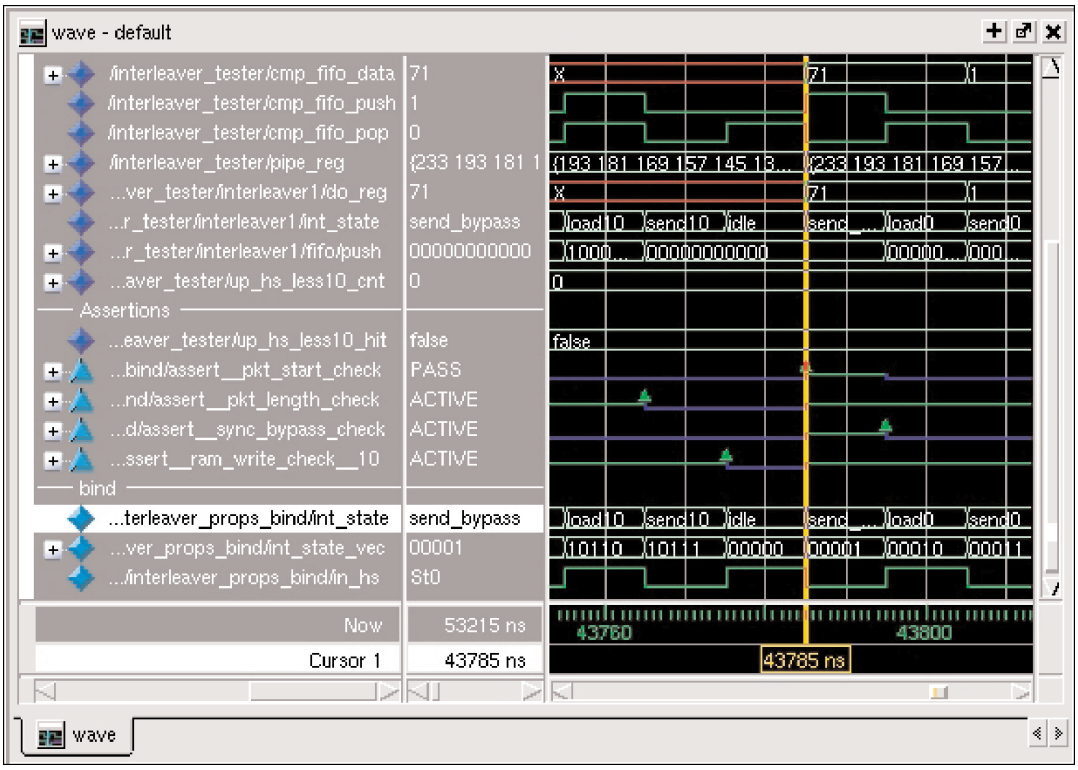

Figure 1. VHDL FSM monitored with SystemVerilog assertions using bind.

Suppose you want to implement functional coverage of the VHDL finite state machine states. With Questa, you can bind any SystemVerilog functionality, such as functional coverage, into a VHDL object:

```
…
covergroup sm_cvg @(posedge clk);
  coverpoint int_state
  {
    bins idle_bin = {idle};
    bins load_bins = {load_bypass, load0, load9, load10};
    bins send_bins = {send_bypass, send0, send9, send10};
    bins others = {wait_idle};
    option.at_least = 500;
  }
  coverpoint in_hs;
  in_hsXint_state: cross in_hs, int_state;
endgroup

sm_cvg sm_cvg_c1 = new;
…
```

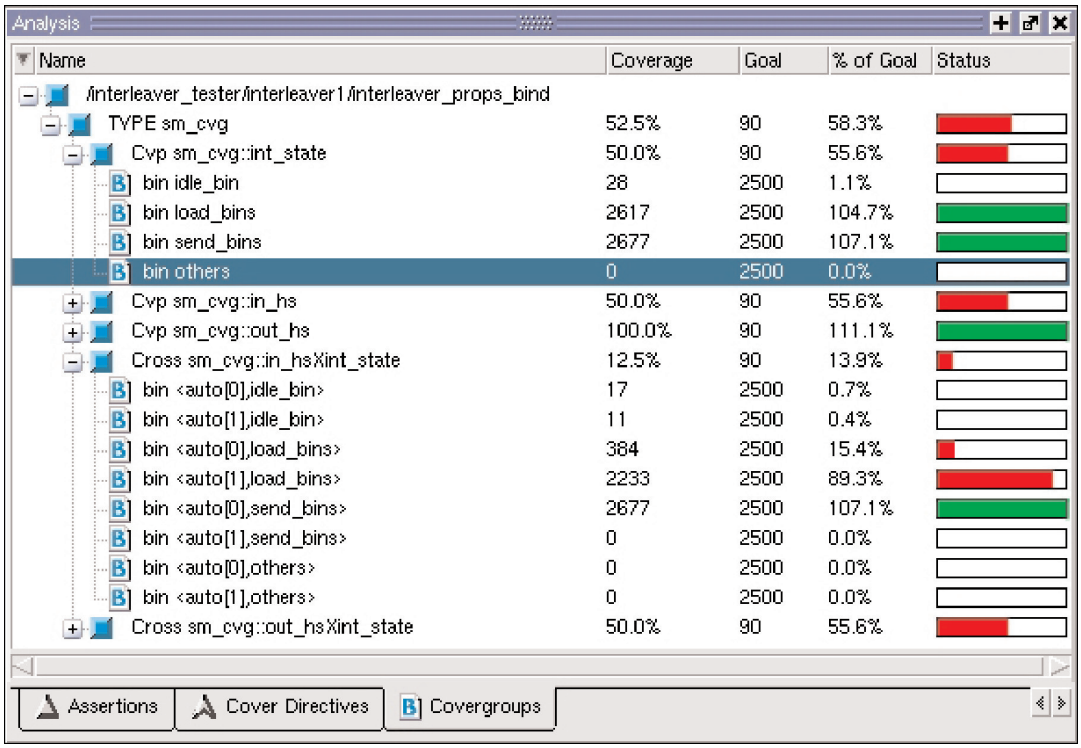

Figure 2. SystemVerilog functional coverage of VHDL using bind.

As with monitoring VHDL components, you create a wrapper to connect the SVA to the VHDL component:

```
module interleaver_binds;
…
// Bind interleaver_props to a specific interleaver instance
// and call this instantiate interleaver_props_bind
bind interleaver_m0 interleaver_props interleaver_props_bind (
    .clk(clk), ..
    .int_state_vec(int_state)
    );

    // connect the SystemVerilog ports to VHDL ports (clk)
    // and to the internal signal (int_state)
…
endmodule
```

Again, you can use either of two options to perform the actual binding in Questa: instantiation or the loading of multiple top modules into the simulator.

```
vsim interleaver_tester interleaver_binds
```

This will load and elaborate both the top-level design and the bind wrapper. As shown below, the SystemVerilog module exists as a sub-module under the bound object.

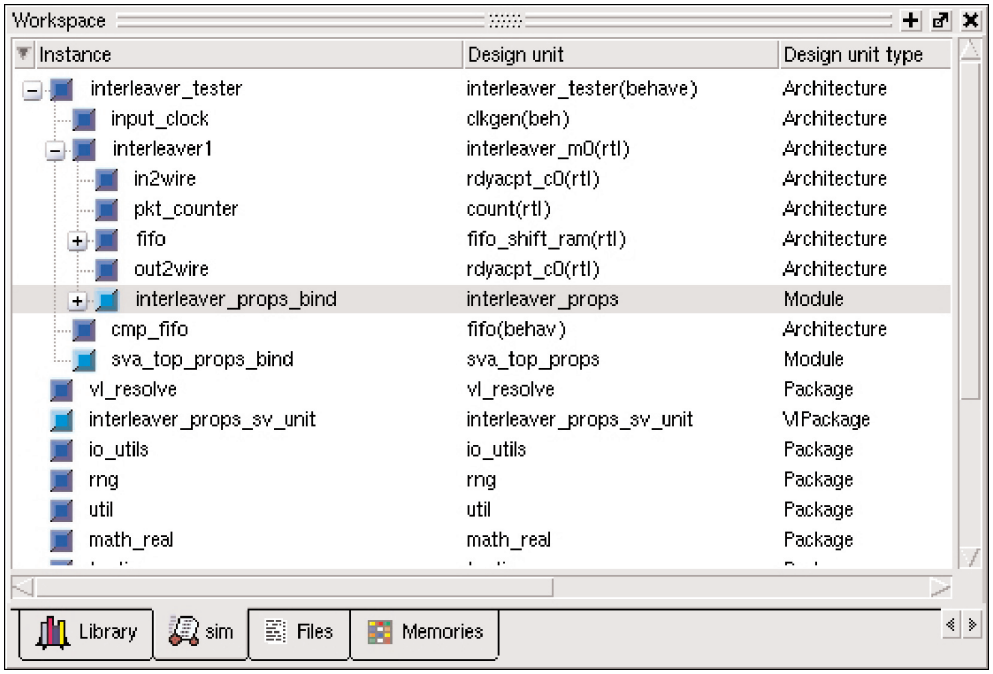

Figure 3. The SystemVerilog module exists as a sub-module under the bound object.

Binding to a VHDL Instance

Questa also supports the binding of SystemVerilog into VHDL design units that are defined by generate or configuration statements.

Consider the following VHDL code:

```
architecture Structure of Test is
  signal A, B, C : std_logic_vector(0 to 3);
…
begin
  TOP          : for i in 0 to 3 generate
    First      : if i = 0 generate
      – configure it..
      for all : thing use entity work.thing(architecture_ONE);
    begin
      Q        : thing port map (A(0), B(0), C(0));
    end generate;

    Second     : for i in 1 to 3 generate
      – configure it..
      for all : thing use entity work.thing(architecture_TWO);
    begin
      Q        : thing port map ( A(i), B(i), C(i) );
    end generate;
  end generate;
end Structure;
```

The following SystemVerilog program defines the assertion:

```
program SVA (input c, a, b);
…
sequence s1;
    @(posedge c) a ##1 b ;
endsequence
cover property (s1);
…
endprogram
```

To tie the SystemVerilog cover directive to the VHDL component, you can use a wrapper module such as the following:

```
module sva_wrapper;
   bind test.top__2.second__1.q   // Bind a specific instance
   SVA                            // to SVA and call this
   sva_bind                       // instantiation sva_bind
   ( .a(A), .b(B), .c(C) );       // Connect the SystemVerilog
                                  // ports to VHDL ports (A, B and C)
endmodule
```

You can instantiate *sva_wrapper* in the top level or simply load multiple top modules into the simulator:

```
vlib work
vlog *.sv
vcom *.vhd
vsim test sva_wrapper
```

This binds the SystemVerilog program, *SVA*, to the specific instance defined by the generate and configuration statements.

You can control the format of generate statement labels by using the *GenerateFormat* variable in the *modelsim.ini* file. See the Questa *User's Manual* for more information.

Hierarchical References

The Questa Signal Spy™ technology provides hierarchical access to bound SystemVerilog objects from VHDL objects. SystemVerilog modules also can access bound VHDL objects using Signal Spy, and they can access bounded Verilog objects using standard Verilog hierarchical references. See the Questa *User's Manual* for more information on the use of Signal Spy.

Summary

Questa gives VHDL users access to the rich set of testbench automation capabilities, native assertions, and functional coverage available in SystemVerilog.

The recommended procedure for binding SystemVerilog and VHDL includes:

- Creating a SystemVerilog program or module to hold the verification code.
- Developing a wrapper module or program block to bind the SystemVerilog code to the VHDL component.
- Loading multiple top-level modules into the simulator or instantiating the SystemVerilog bind wrapper.

For more information, call us or visit: www.mentor.com

Corporate Headquarters
Mentor Graphics Corporation
8005 S.W. Boeckman Road
Wilsonville, Oregon 97070USA
Phone: 503-685-7000
North American Support Center
Phone: 800-547-4303
Fax: 800-684-1795

Silicon Valley
Mentor Graphics Corporation
1001 Ridder Park Drive
San Jose, California 95131 USA
Phone: 408-436-1500
Fax: 408-436-1501

Europe
Mentor Graphics
Deutschland GmbH
Arnulfstrasse 201
80634 Munich
Germany
Phone: +49.89.57096.0
Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics Taiwan
Room 1603, 16F,
International Trade Building
No. 333, Section 1, Keelung Road
Taipei, Taiwan, ROC
Phone: 886-2-27576020
Fax: 886-2-27576027

Japan
Mentor Graphics Japan Co., Ltd.
Gotenyama Hills
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140
Japan
Phone: 81-3-5488-3030
Fax: 81-3-5488-3031

MGC 7-05 TECH6690-w