SVA Advanced Topics: SVAUnit and Assertions for Formal (part 1)

Andra Radu

Ionuț Ciocîrlan

AMIQ Consulting
SVAUnit tutorial topics

• Introduction to SystemVerilog Assertions (SVAs)

• Planning SVA development

• Implementation

• SVA verification using SVAUnit

• SVA test patterns
Introduction to SystemVerilog Assertions (SVAs)
Assertions and properties

• What is an assertion?

```haskell
assert (a |-> b)
else $error("Assertion failed!")
```

• What is a property?

```haskell
property p_example;
  a |-> b
endproperty
```
Simple assertion example

After the rise of request signal, the acknowledge signal should be asserted no later than 3 clocks cycles.

```verbatim
property req_to_rise_p;
    @(posedge clk)
    $rose(req) |-> ##[1:3] $rose(ack);
endproperty

ASSERT_LABEL: assert property (req_to_rise_p)
else `uvm_error("ERR", "Assertion failed")
```
Types of SystemVerilog Assertions

- Immediate

\[
\text{assert (expression) pass_statement} \\
[\text{else fail_statement}]
\]

- Concurrent

After the assertion of request signal, the acknowledge signal should be asserted no later than 3 clock cycles.

- SVA succeeded
- SVA failed
Assertions are used

• In a verification component

• In a formal proof kit

• In RTL generation
  “Revisiting Regular Expressions in SyntHorus2: from PSL SEREs to Hardware” (Fatemeh (Negin) Javaheri, Katell Morin-Allory, Dominique Borrione)

• For test patterns generation
  “Towards a Toolchain for Assertion-Driven Test Sequence Generation” (Laurence PIERRE)
SVAs advantages

• Fast
• Non-intrusive
• Flexible
• Coverable
Planning SVA development
Identify design characteristics

- Defined in a document (design specification)
- Known or specified by the designer
- The most common format is of the form *cause and effect*: antecedent $\rightarrow$ consequent

- Antecedent: $\texttt{rose(req)}$

- Consequent: $\texttt{#[1:3] \: \texttt{rose(ack)}}$
Complex assertions are typically constructed from complex sequences and properties.

\[
a \#1 \ b[*1:2] \Rightarrow c \#1 \ d[*1:2] \Rightarrow \text{fell}(a)
\]

\[
\text{sequence} \ \text{seq}(\text{arg1}, \text{arg2});
\text{arg1} \#1 \text{arg2}[*1:2];
\text{endsequence}
\]

\[
\text{seq}(a, b) \Rightarrow \text{seq}(c, d) \Rightarrow \text{fell}(a)
\]
Implementation
Coding guidelines

- Avoid duplicating design logic in assertions
- Avoid infinite assertions
- Reset considerations
- Mind the sampling clock
Coding guidelines (contd.)

- Always check for unknown condition (‘X’)
- Assertion naming
- Detailed assertion messages
- Assertion encapsulation
Best practices

• Review the SVA with the designer to avoid DS misinterpretation

• Use strong in assertions that may never complete:

```
assert property ( req |-> strong(##[1:$] ack));
```

• Properties should not hold under certain conditions (reset, enable switch)

```
assert property
  @(posedge clk) disable iff (!setup || !rst_n)
    req |-> strong(##[1:$] ack)
);
```
• Avoid overlapping assertions that contradict each other

✓ CPU_0: \[
\text{assert property (WRITE } \implies \text{ERROR)};
\]

✓ CPU_1: \[
\text{assert property (WRITE } \implies \text{!ERROR)};
\]

assert property (WRITE and CPU==0 } \implies \text{ERROR)};

assert property (WRITE and CPU==1 } \implies \text{!ERROR)};
Best practices (contd.)

• Use the $sampled() function in action blocks

```verilog
assert property (@(posedge clk) ack == 0 )
else
  `uvm_error("ERROR", $sformatf("Assertion failed. ack is %d", $sampled(ack)));
```
Assertion example

- AMBA APB protocol specification:

The bus only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock.
Assertion example (contd.)

• Antecedent (the SETUP phase)

```verilog
sequence setup_phase_s;
    $rose(psel) and $rose(pwrite) and (!penable) and (!pready);
endsequence
```

• Consequent (the ACCESS phase)

```verilog
sequence access_phase_s;
    $rose(penable) and $rose(pready) and $stable(pwrite) and $stable(pwdata) and $stable(paddr) and $stable(psel);
endsequence
```
• The property can be expressed as:

```verilog
property access_to_setup_p;
    @(posedge clk) disable iff (reset)
        setup_phase_s |=> access_phase_s;
endproperty
```

• The assertion will look like:

```verilog
assert property (access_to_setup_p)
else `uvm_error("ERR", "Assertion failed")
```
Does it work as intended?
SVA Verification with SVAUnit
SVA Verification Challenges

Easy to:
- Update
- Enhance
- Disable

Clear separation between validation and SVA definition code

Results should be:
- Deterministic
- Repeatable
Introducing SVAUnit

- Structured framework for Unit Testing for SVAs
- Allows the user to decouple the SVA definition from its validation code
- UVM compliant package written in SystemVerilog
- Encapsulate each SVA testing scenario inside an unit test
- Easily controlled and supervised using a simple API
SVAUnit Infrastructure

- **SVAUnit Testbench**
  - Enables SVAUnit
  - Instantiates SVA interface
  - Starts test

- **SVAUnit Test**
  - Contains the SVA scenario

- **SVAUnit Test Suite**
  - Test and test suite container
Example specification

- AMBA APB protocol specification:

The bus only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock.
interface apb_if (input pclk);
  logic psel;
  logic pwrite;
  logic penable;
  logic [`ADDR_WIDTH-1 :0] paddr;
  logic [`WDATA_WIDTH-1:0] pwdata;
endinterface

APB sequences definitions

APB property definition

APB assertion definition

endinterface
• Antecedent (the SETUP phase)

sequence setup_phase_s;
   $rose(psel) and $rose(pwrite)
   and (!penable) and (!pready);
endsequence

• Consequent (the ACCESS phase)

sequence access_phase_s;
   $rose(penable) and $rose(pready) and
   $stable(pwrite) and $stable(pwdata) and
   $stable(paddr) and $stable(psel);
endsequence
APB property & assertion definitions

• The property can be expressed as:

```verilog
property access_to_setup_p;
  @(posedge clk) disable iff (reset)
  setup_phase_s |= access_phase_s;
endproperty
```

• The assertion will look like:

```verilog
assert property (access_to_setup_p)
else `uvm_error("ERR", "Assertion failed")
```
Example of SVAUnit Testbench

```verilog
module top;
  // Instantiate the SVAUnit framework
  `SVAUNIT_UTILS

  // APB interface with the SVA we want to test
  apb_if an_apb_if(.clk(clock));

  initial begin
    // Register interface with the uvm_config_db
    uvm_config_db#(virtual an_if)::
      set(uvm_root::get(), "*", "VIF", an_apb_if);

    // Start the scenarios
    run_test();
  end

  endmodule
```

![Diagram of SVAUnit Testbench](image)
Example of SVAUnit Test

class utl extends svaunit_test;
    // The virtual interface used to drive the signals
    virtual apb_if apb_vif;

    function void build_phase(input uvm phase phase);
        // Retrieve the interface handle from the uvm_config_db
        if (!uvm_config_db#(virtual an_if)::get(this, "", "VIF", apb_vif))
            `uvm_fatal("UT1_NO_VIF_ERR", "SVA interface is not set!"))
            // Test will run by default;
            disable_test();
    endfunction

    task test();
        // Initialize signals
        // Create scenarios for SVA verification
    endtask
endclass
APB – SVAUnit test steps

1. Enable the APB SVA
2. Initialize the interface signals
3. Generate setup phase stimuli
4. Generate access phase stimuli
5. SVA checks based on generated stimuli
Enable SVA and initialize signals

...
Generate setup phase stimuli

... task generate_setup_phase_stimuli(bit valid);
...

// Stimuli for valid SVA scenario
valid == 1 ->
pwrite == 1 && psel == 1 && penable == 0 && pready == 0;

// Stimuli for invalid SVA scenario
valid == 0 ->
pwrite != 1 || psel != 1 || penable != 0 || pready != 0;

... endtask

...
Generate access phase stimuli

... task generate_access_phase_stimuli(bit valid); ...

// Constrained stimuli for valid SVA scenario
valid == 1 ->
pwdata == apb_vif.pwdata && paddr == apb_vif.paddr &&
pwrite == 1 && psel == 1 && penable == 1 && pready == 1;

// Constrained stimuli for invalid SVA scenario
valid == 0 ->
pwdata != apb_vif.pwdata || paddr != apb_vif.paddr ||
pwrite != 1 || psel != 1 || penable != 1 || pready != 1;

... endtask ...
...
... 

\[
\text{if (valid\_setup\_phase) if (valid\_access\_phase) vpiw.fail\_if\_sva\_not\_succeeded("APB\_PHASES", "The assertion should have succeeded");}
\]

\[
\text{else vpiw.fail\_if\_sva\_succeeded("APB\_PHASES", "The assertion should have failed");}
\]

\[
\text{else vpiw.pass\_if\_sva\_not\_started("APB\_PHASES", "The assertion should not have started");}
\]

...
Example of SVAUnit Test Suite

class uts extends svaunit_test_suite;
    // Instantiate the SVAUnit tests
    utl utl;
    ...
    ut10 ut10;

    function void build_phase(input uvm_phase phase);
        // Create the tests using UVM factory
        utl = utl::type_id::create("utl", this);
        ...
        ut10 = ut10::type_id::create("ut10", this);

        // Register tests in suite
        `add_test(utl);
        ...
        `add_test(ut10);
    endfunction

dendclass
SVAUnit Test API

**CONTROL**
- disable_all_assertions();
- enable_assertion(sva_name);
- enable_all_assertions();
  ...

**CHECK**
- fail_if_sva_does_not_exists(sva_name, error_msg);
- pass_if_sva_not_succeeded(sva_name, error_msg);
- pass/fail_if(expression, error_msg);
  ...

**REPORT**
- print_status();
- print_sva();
- print_report();
  ...
SVAUnit Flow

Create SVAUnit Testbench
Create an SVAUnit Test
Implement test() task

Create an SVAUnit Test Suite
Instantiate test in Test Suite
Register tests in test suite

Simulate
Scan report
Error reporting

Name of SVAUnit check

SVAUnit test path

Name of SVA under test

Custom error message

UVM_ERROR @ 55000 ns [SVAUNIT_FAIL_IF_SVA_SUCCEEDED_ERR]: [x_z_suite.addr_x_z_test::x_z_addr_ut
AMIQ_APB_ILLEGAL_ADDR_VALUE_ERR] The assertion should have failed
Hierarchy report

UVM_INFO @ 56000 ns [protocol_ts]:
  protocol_ts
    protocol_ts.protocol_test1
    protocol_ts.protocol_test2
    protocol_ts.x_z_suite
      x_z_suite.addr_x_z_test
      x_z_suite.slverr_x_z_test
      x_z_suite.sel_x_z_test
      x_z_suite.write_x_z_test
      x_z_suite.strb_x_z_test
      x_z_suite.prot_x_z_test
      x_z_suite.enable_x_z_test
      x_z_suite.ready_x_z_test
Test scenarios exercised

---

protocol_ts test suite: Status statistics

* protocol_ts FAIL (2/3 test cases PASSED)
* protocol_ts.x_z_suite FAIL (0/8 test cases PASSED)
protocol_ts.protocol_test2 PASS (13/13 assertions PASSED)
protocol_ts.protocol_test1 PASS (13/13 assertions PASSED)

UVM_INFO @ 56000 ns [protocol_ts]:

3/3 Tests ran during simulation

protocol_ts.x_z_suite
protocol_ts.protocol_test2
protocol_ts.protocol_test1
SVAs and checks exercised

---

protocol_ts test suite : SVA and checks statistics ---

AMIQ_APB_ILLEGAL_SEL_TRANSITION_TR_PHASES_ERR  13/13 checks PASSED  
  SVAUNIT_FAIL_IF_SVA_SUCCEEDED_ERR 1/1 times PASSED  
  SVAUNIT_FAIL_IF_SVA_NOT_SUCCEEDED_ERR 2/2 times PASSED  
  SVAUNIT_FAIL_IF_SVA_DOES_NOT_EXISTS_ERR 7/7 times PASSED  
  SVAUNIT_PASS_IF_SVA_IS_ENABLE_ERR 3/3 times PASSED

AMIQ_APB_ILLEGAL_SEL_TRANSITION_DURING_TRANSFER_ERR  13/13 checks PASSED  
  SVAUNIT_FAIL_IF_SVA_NOT_SUCCEEDED_ERR 1/1 times PASSED  
  SVAUNIT_FAIL_IF_SVA_SUCCEEDED_ERR 2/2 times PASSED  
  SVAUNIT_FAIL_IF_SVA_DOES_NOT_EXISTS_ERR 7/7 times PASSED  
  SVAUNIT_PASS_IF_SVA_IS_ENABLE_ERR 3/3 times PASSED
SVA test patterns
Simple implication test

- \( a \ and \ b \ |\Rightarrow \ c \)

```vhdl
repeat (test_loop_count) begin
    randomize(stimuli_for_a, stimuli_for_b, stimuli_for_c);

    interface.a <= stimuli_for_a;
    interface.b <= stimuli_for_b;
    @(posedge an_vif.clk);

    interface.c <= stimuli_for_c;
    @(posedge interface.clk);

    @(posedge interface.clk);
    if (stimuli_for_a == 1 && stimuli_for_b == 1)
        if (stimuli_for_c == 1)
            vpiw.fail_if_sva_not_succeeded("IMPLICATION_ASSERT",
                "The assertion should have succeeded!");
        else
            vpiw.fail_if_sva_succeeded("IMPLICATION_ASSERT",
                "The assertion should have failed!");
        else
            vpiw.pass_if_sva_not_started("IMPLICATION_ASSERT",
                "The assertion should not have started!");
    end
```
Multi-thread antecedent/consequent

- $\$rose(a) ##[1:4] b \rightarrow ##[1:3] c$

```vhdl
repeat (test_loop_count) begin
  // Generate valid delays for asserting b and c signals
  randomize(delay_for_b inside {[1:4]}, delay_for_c inside {[1:3]});
  interface.a <= 1;

  repeat (delay_for_b)
    @(posedge interface.clk);
  interface.b <= 1;

  vpiw.pass_if_sva_started_but_not_finished("MULTITHREAD_ASSERT",
      "The assertion should have started but not finished!");

  repeat (delay_for_c)
    @(posedge interface.clk);
  interface.c <= 1;

  vpiw.pass_if_sva_succeeded("MULTITHREAD_ASSERT",
      "The assertion should have succeeded!");
end
```
Multi-thread antecedent/consequent (contd.)

• \$rose(a) ##[1:4] b \rightarrow ##[1:3] c

```verilog
repeat (test_loop_count) begin
  // Generate invalid delays for asserting b and c signals
  randomize(delay_for_b inside {0:10}, delay_for_c inside {0, [4:10]});
  interface.a <= 1;

  repeat (delay_for_b)
    @(posedge interface.clk);
  interface.b <= 1;

  vpiw.pass_if_sva_not_succeeded("MULTITHREAD_ASSERT",
                                "The assertion should have failed!");

  repeat (delay_for_c)
    @(posedge interface.clk);
  interface.c <= 1;

  if (delay_for_b < 5)
    vpiw.fail_if_sva_succeeded("MULTITHREAD_ASSERT",
                               "The assertion should have failed!");
end
```
Consecutive repetition

- \( a \rightarrow b[*1:2] \#\#1 \ c \)

```vhdl
repeat (test_loop_count) begin
    randomize(stimuli_for_a, stimuli_for_c, number_of_b_cycles <= 2);

    interface.a <= stimuli_for_a;

    repeat (number_of_b_cycles) begin
        randomize(stimuli_for_b)
        interface.b <= stimuli_for_b;
        if (stimuli_for_b == 1) number_of_b_assertions += 1;

        @(posedge interface.clk);
    end

    if (stimuli_for_a == 1 && number_of_b_assertions == number_of_b_cycles &&
        number_of_b_assertions > 0) vpiw.pass_if_sva_started_but_not_finished("IMPLICATION_ASSERT", "The assertion should have started but not finished!");
    @(posedge interface.clk);

... // (continued on the next slide)
```
Consecutive repetition (contd.)

- $a \implies b[*1:2] \#1 c$

```vhdl
...  
// (continued from previous slide)

interface.c <= stimuli_for_c;
@ (posedge interface.clk);

if (stimuli_for_a == 1)
    if (number_of_b_assertions != number_of_b_cycles ||
        number_of_b_assertions == 0 ||
        stimuli_for_c == 0)
        vpiw.fail_if_sva_succeeded("IMPLICATION_ASSERT",
                               "The assertion should have failed!");
    else
        vpiw.fail_if_sva_not_succeeded("IMPLICATION_ASSERT",
                                      "The assertion should have succeeded!");

end // end of test repeat loop
```
• \( a \rightarrow b[*0:2] \#\#1 \) \( c \)

```verilog
repeat (test_loop_count) begin
    randomize(stimuli_for_a, stimuli_for_c, number_of_b_cycles <= 2);

    interface.a <= stimuli_for_a;

    repeat (number_of_b_cycles) begin
        randomize(stimuli_for_b)
        interface.b <= stimuli_for_b;
        if (stimuli_for_b == 1) number_of_bAssertions += 1;

        @(posedge interface.clk);
    end

    if (stimuli_for_a == 1 && number_of_bAssertions == number_of_b_cycles)
        && number_of_bAssertions > 0)
    vpiw.pass_if_sva_started_but_not_finished("IMPLICATION_ASSERT",
        "The assertion should have started but not finished!");
    @(posedge interface.clk);

... // (continued on the next slide)
```
Repetition range with zero (contd.)

• $a \Rightarrow b[0:2] \#1 \ c$

```vhdl
// (continued from previous slide)

interface.c <= stimuli_for_c;
@(posedge interface.clk);

if (stimuli_for_a == 1)
  if (number_of_b_assertions != number_of_b_cycles ||
      number_of_b_assertions == 0 ||
      stimuli_for_c == 0)
    vpiw.fail_if_sva_succeeded("REPETITION_RANGE0_ASSERT",
      "The assertion should have failed!");
  else
    vpiw.fail_if_sva_not_succeeded("REPETITION_RANGE0_ASSERT",
      "The assertion should have succeeded!");

end // end of test repeat loop
```
Sequence disjunction

- \( a \mapsto (b \bowtie\\^1 c) \text{ or } (d \bowtie\\^1 e) \)

```verilog
repeat (test_loop_count) begin
    randomize(stimuli_for_a, stimuli_for_b, stimuli_for_c, stimuli_for_d, stimuli_for_e);

    interface.a <= stimuli_for_a;
    @(posedge interface.clk);
    fork
        begin
            Stimuli for branch: (b \bowtie\\^1 c)
            SVA state check based on branch stimuli
        end
        begin
            Stimuli for branch: (c \bowtie\\^1 d)
            SVA state check based on branch stimuli
        end
    end
end
```
Sequence disjunction (contd.)

- \( a \mid=> (b \#\#1 c) \textbf{or} (d \#\#1 e) \)

```vhdl
...

// Stimuli for branch (b #1 c)
fork
  begin
    interface.b <= stimuli_for_b;
    @(posedge interface.clk);
    interface.c <= stimuli_for_c;
    @(posedge interface.clk);
    @(posedge interface.clk);
    // SVA state check based on branch stimuli
    sva_check_phase(interface.a, interface.b, interface.c);
  end
join
```
• $a |=> (b ##1 c) \text{ or } (d ##1 e)$

... // Stimuli for branch (d ##1 e)
fork
    begin
        interface.b <= stimuli_for_d;
        @(posedge interface.clk);
        
        interface.c <= stimuli_for_e;
        @(posedge interface.clk);
        
        @(posedge interface.clk);
        // SVA state check based on branch stimuli
        sva_check_phase(interface.a, interface.d, interface.e);
    end
join
Sequence disjunction (contd.)

- \( a \implies (b \equiv_1 c) ~ or ~ (d \equiv_1 e) \)

// SVA state checking task used in each fork branch

task sva_check_phase(bit stimuli_a, bit stimuli_b, bit stimuli_c);
  if (stimuli_a)
    if (stimuli_b && stimuli_c)
      vpiw.pass_if_sva_succeeded("DISJUNCTION_ASSERT",
      "The assertion should have succeeded");
  else
    vpiw.fail_if_sva_succeeded("DISJUNCTION_ASSERT",
      "The assertion should have failed");
endtask
Tools integration

Simulator independent!
Availability

- SVAUnit is an open-source package released by AMIQ Consulting
- We provide:
  - SystemVerilog and simulator integration codes
  - AMBA-APB assertion package
  - Code templates and examples
  - HTML documentation for API

https://github.com/amiq-consulting/svaunit
Conclusions

- SVAUnit decouples the checking logic from SVA definition code
- Safety net for eventual code refactoring
- Can also be used as self-checking documentation on how SVAs work
- Quick learning curve
- Easy-to-use and flexible API
- Speed up verification closure
- Boost verification quality
Thank you!