SystemVerilog Assertions Verification

AMIQ Consulting

Ionuț Ciocîrlan   Andra Radu

accellera SYSTEMS INITIATIVE

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

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

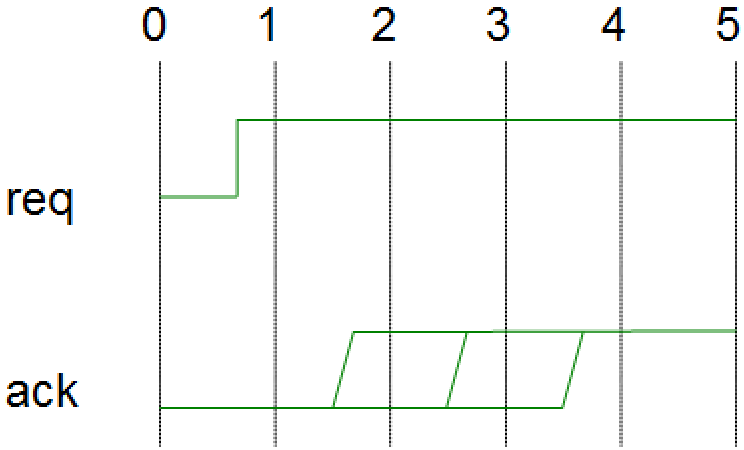
- What is an assertion?

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

- What is a property?

```
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.

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

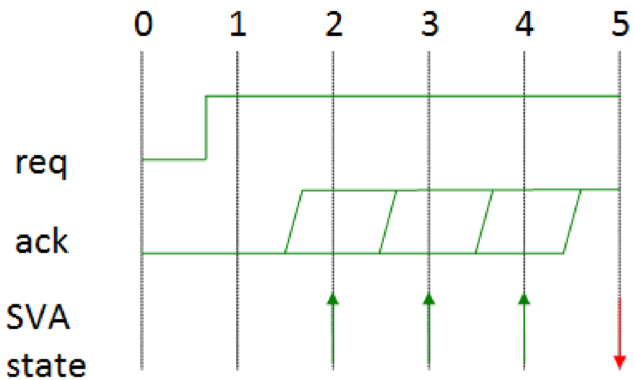
```
ASSERT_LABEL: assert property (req_to_rise_p)
else `uvm_error(“ERR", "Assertion failed")
```

Types of SystemVerilog Assertions

- Immediate

```
assert (expression) pass_statement
[else fail_statement]
```

- Concurrent



After the assertion of request signal, the acknowledge signal should be asserted no later than 3 clocks 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-instrusive
- 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 |-> consequent

- Antecedent:

```
$rose(req)
```

- Consequent:

```
##[1:3] $rose(ack)
```

Keep it simple. Partition!

- Complex assertions are typically constructed from complex sequences and properties.

```
a ##1 b[*1:2] |=> c ##1 d[*1:2] |=> $fell(a)
```

```
sequence seq(arg1, arg2);
 arg1 ##1 arg2[*1:2];
endsequence
```

```
seq(a, b) |=> seq(c, d) |=> $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:

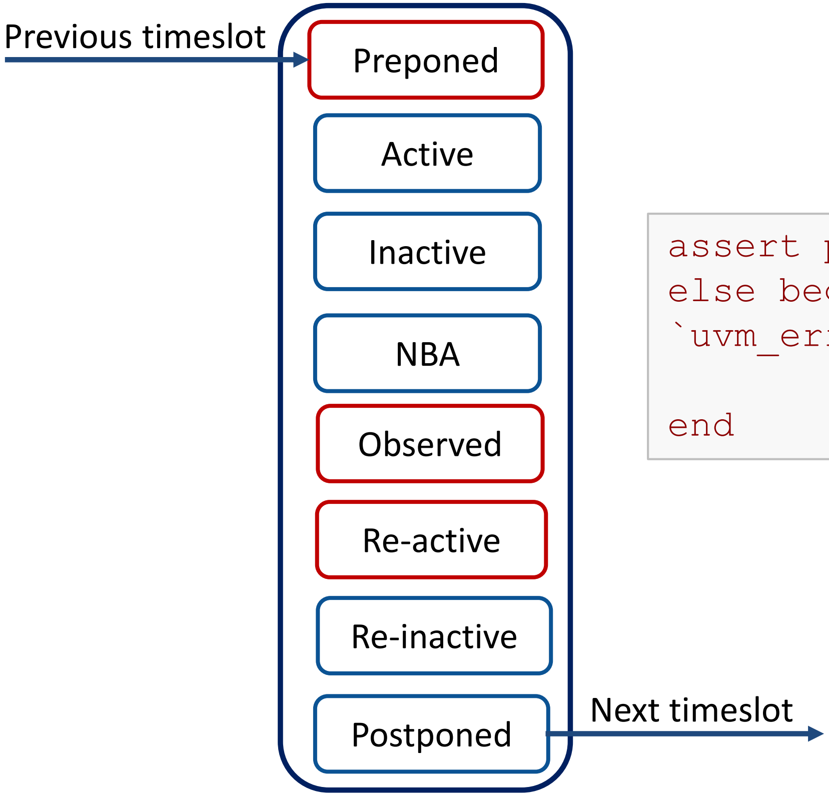
```
assert property ( req |-> strong(##[1:$] ack));
```

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

```
assert property (
@(posedge clk) disable iff (!setup || reset)
      req |-> strong(##[1:3] ack)
);
```

Best practices (contd.)

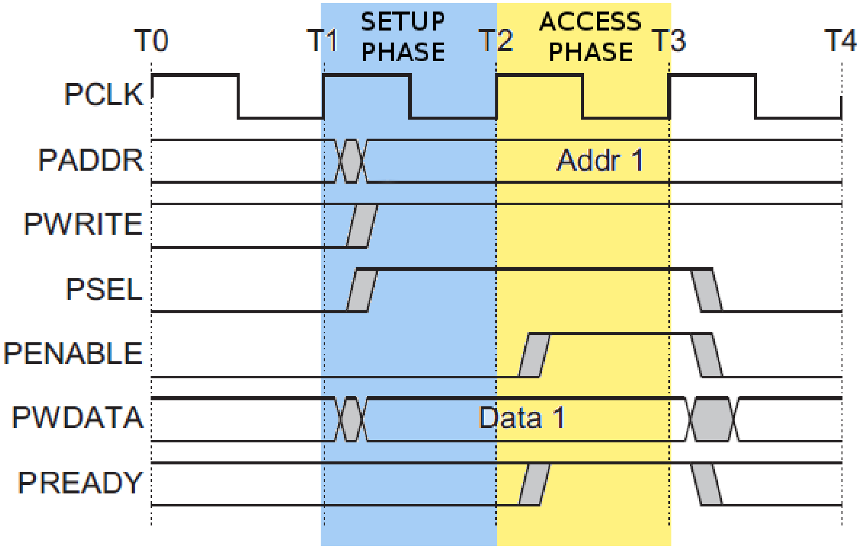
- Use the $sampled() function in action blocks



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

Assertion example

- AMBA APB protocol specification:

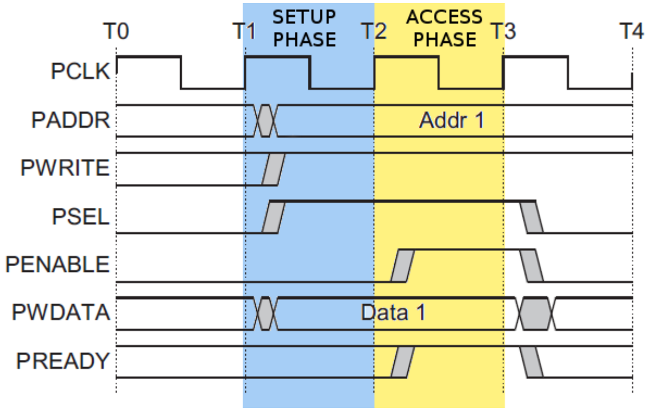


The bus only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock.

Assertion example (contd.)

- Antecedent (the SETUP phase)

```
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
```

Assertion example (contd.)

- The property can be expressed as:

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

- The assertion will look like:

```
assert property (access_to_setup_p)
else `uvm_error("ERR", "Assertion failed")
```

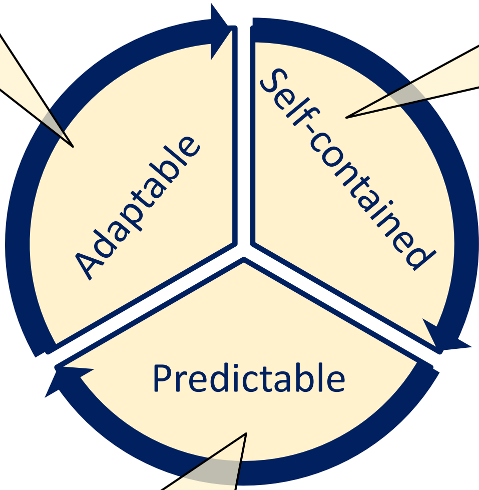
Does it work as intended?

SVA Verification with SVAUnit

SVA Verification Challenges

Easy to:
- Update
- Enhance
- Disable

Clear separation between validation and SVA definition code

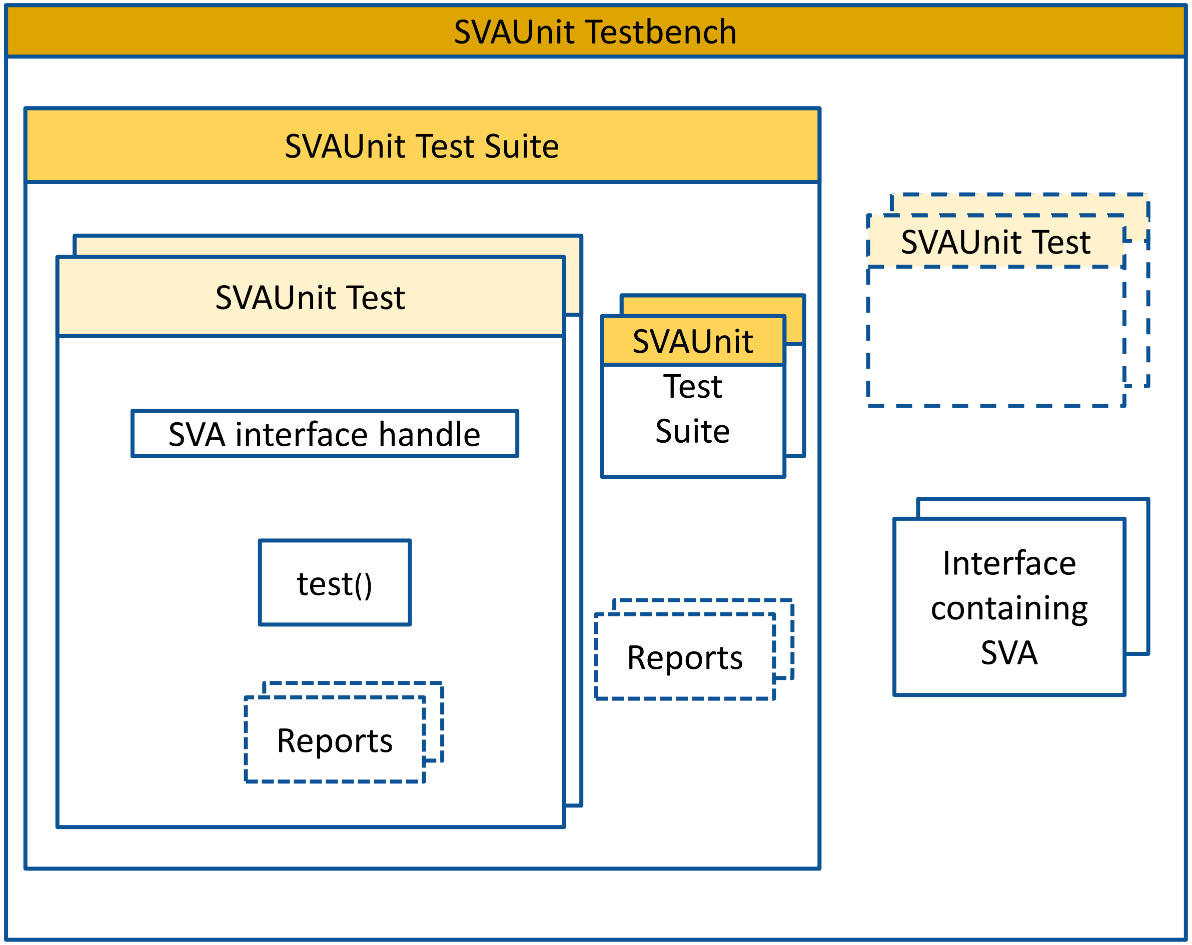


Results should be:
- Deterministic
- Repeatable

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

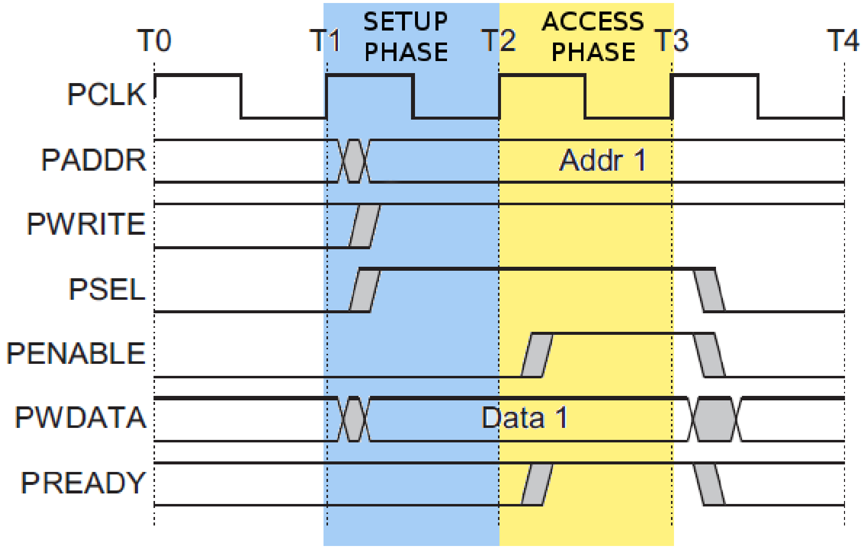
SVAUnit Infrastructure



- **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

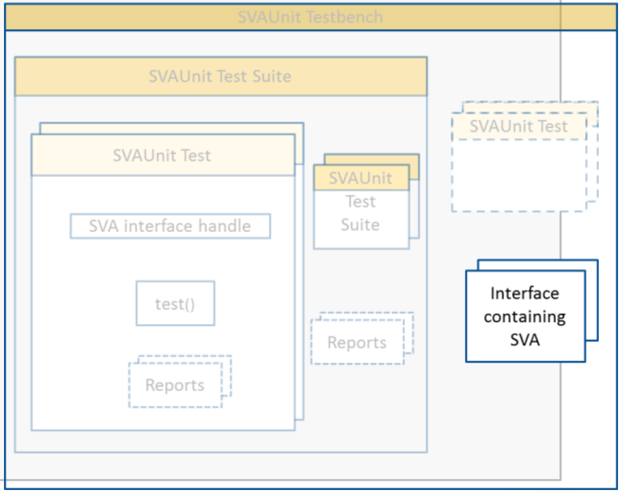
- AMBA APB protocol specification:



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.

Example APB interface

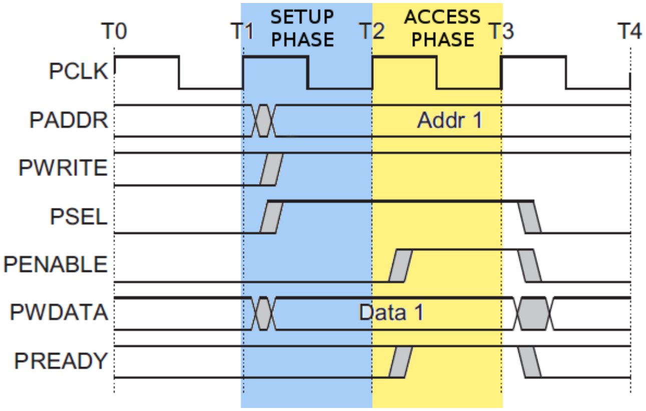
```
interface apb_if (input pclk);
  logic                         psel;
  logic                         pwrite;
  logic                         penable;
  logic                         pready;
  logic [`ADDR_WIDTH-1 :0] paddr;
  logic [`WDATA_WIDTH-1:0] pwdata;

    APB sequences definitions

    APB property definition

    APB assertion definition

endinterface
```



APB sequences definitions

- 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:

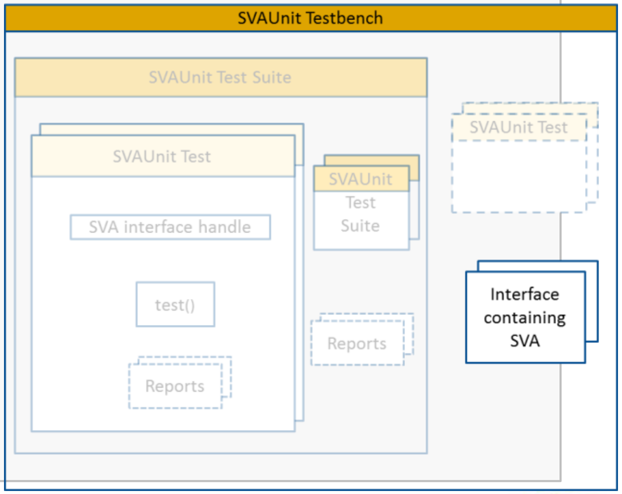
```
property access_to_setup_p;
  @(posedge clk) disable iff (reset)
  setup_phase_s |=> access_phase_s;
endproperty
```

- The assertion will look like:

```
assert property (access_to_setup_p)
else `uvm_error("ERR", "Assertion failed")
```

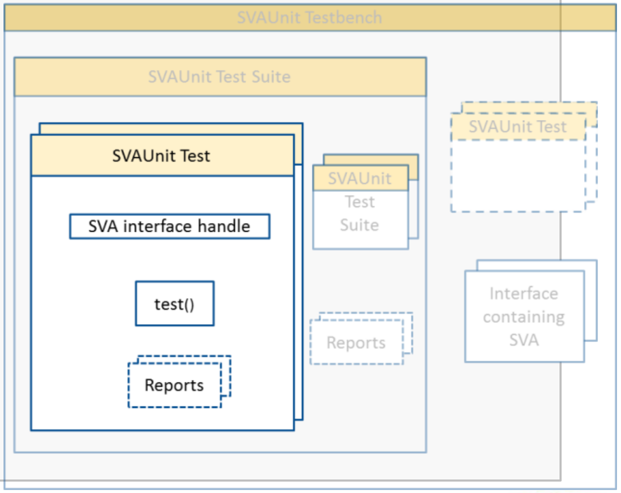
Example of SVAUnit Testbench

```
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
```

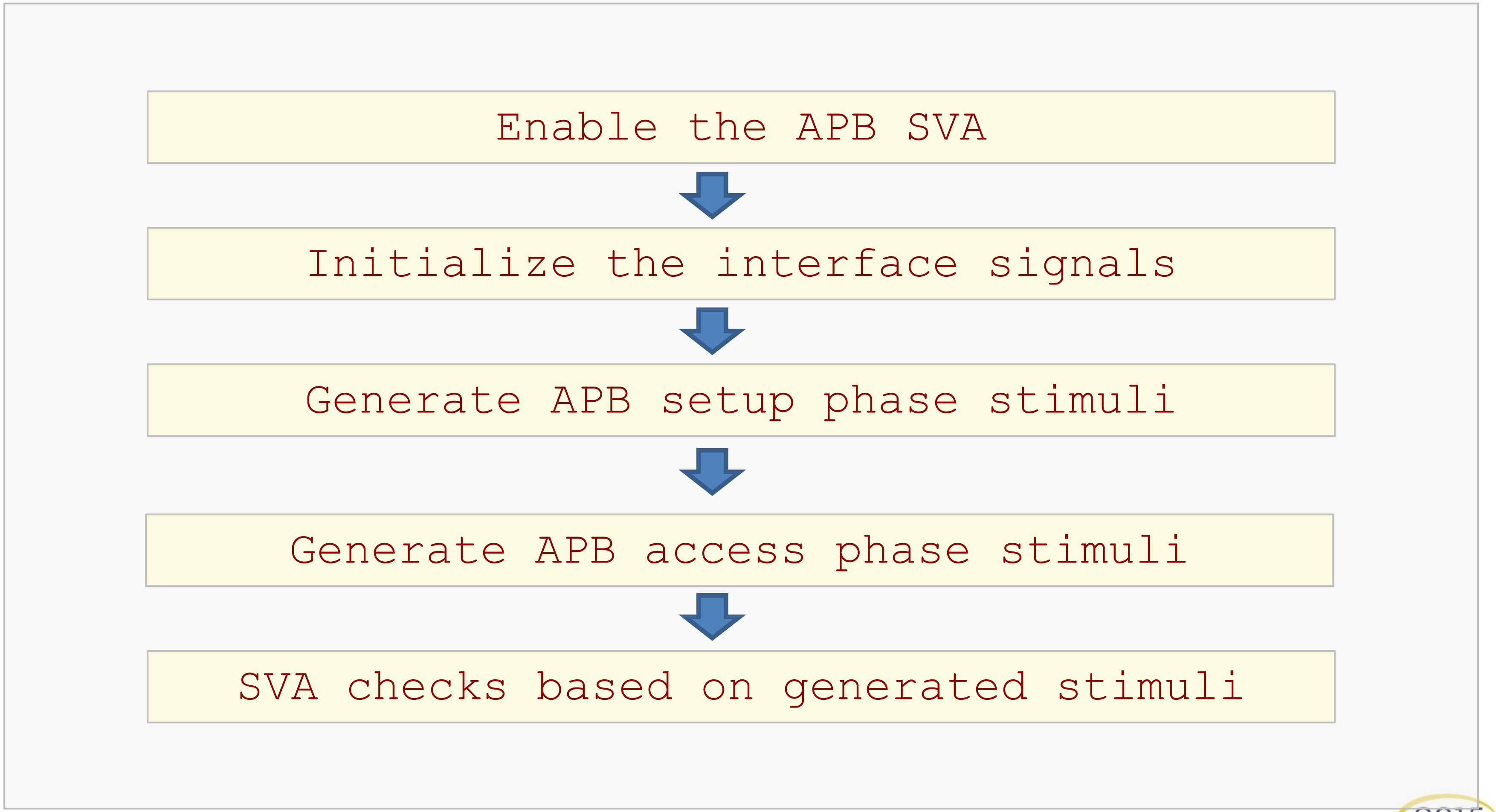


Example of SVAUnit Test

```
class ut1 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



Enable the APB SVA

Initialize the interface signals

Generate APB setup phase stimuli

Generate APB access phase stimuli

SVA checks based on generated stimuli

Enable SVA and initialize signals

```
...

 // Enable the APB SVA
  vpiw.disable_all_assertions();
  vpiw.enable_assertion("APB_PHASES");

  // Initialize signals
  task initialize_signals();
    apb_vif.addr    <= 32'b0;
    apb_vif.wdata   <= 32'b0;
    apb_vif.write   <=  1'b0;
    apb_vif.enable  <=  1'b0;
    apb_vif.sel     <=  1'b0;
  endtask

...
```

Generate APB setup phase stimuli

```
...

  task generate_setup_phase_stimuli(bit valid);
    ...
    // Stimuli for valid SVA scenario
    valid == 1 ->
    write == 1 && sel == 1 && enable == 0 && ready == 0;

    // Stimuli for invalid SVA scenario
    valid == 0 ->
    write != 1 || sel != 1 || enable != 0 || ready != 0;


    ...
  endtask


...
```

Generate APB access phase stimuli

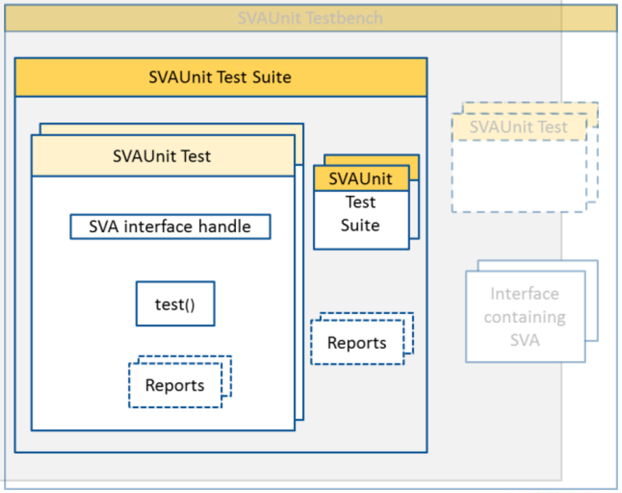
```
...

  task generate_access_phase_stimuli(bit valid);
    ...

     // Constrained stimuli for valid SVA scenario
     valid == 1  ->
     wdata == apb_vif.wdata && addr == apb_vif.addr &&
     write == 1 && sel == 1 && enable == 1 && ready == 1;


    // Constrained stimuli for invalid SVA scenario
     valid == 0 -> wdata != apb_vif.wdata ||  addr != apb_vif.addr ||
     write != 1 || sel != 1 || enable != 1 || ready != 1;

    ...
  endtask
...
```

SVA state checking

```
...

 if (valid_setup_phase)
   if (valid_access_phase)
     vpiw.fail_if_sva_not_succeeded("APB_PHASES",
          "The assertion should have succeeded!");
     else
     vpiw.fail_if_sva_succeeded("APB_PHASES",
          "The assertion should have failed!");
 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
   ut1 ut1;
   ...
   ut10 ut10;

   function void build_phase(input uvm_phase phase);
     // Create the tests using UVM factory
     ut1 = ut1::type_id::create("ut1", this);
     ...
     ut10 = ut10::type_id::create("ut10", this);

     // Register tests in suite
     `add_test(ut1);
     ...
     `add_test(ut10);
   endfunction

endclass
```



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);

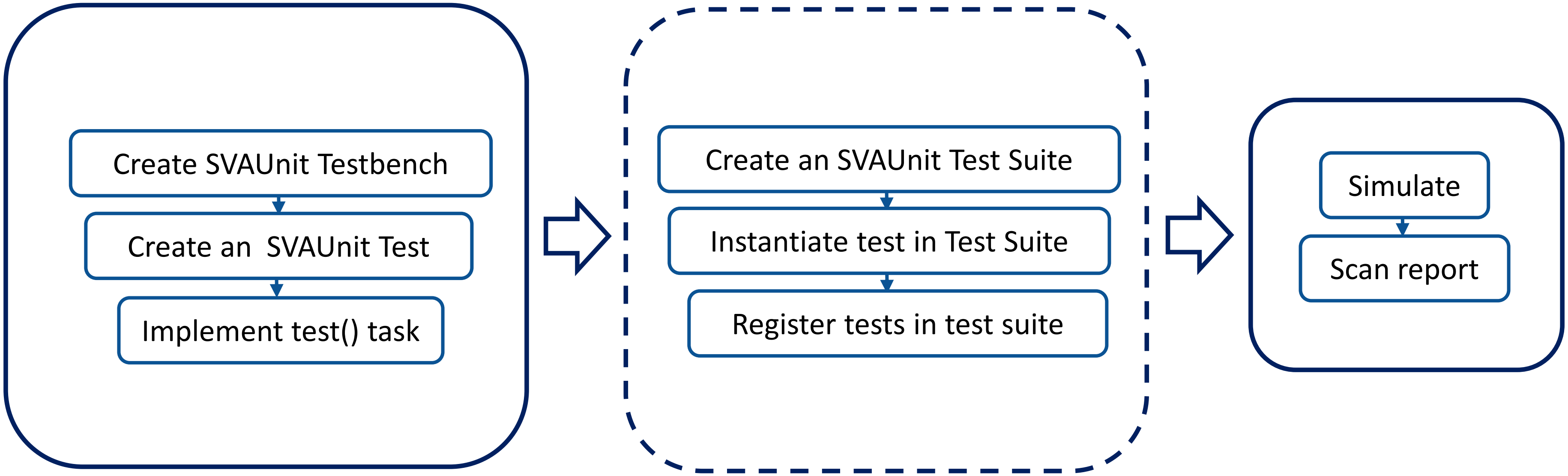
. . .

REPORT

- print_status();
- print_sva();
- print_report();

. . .

SVAUnit Flow



Create SVAUnit Testbench

Create an SVAUnit Test

Implement test() task

Create an SVAUnit Test Suite

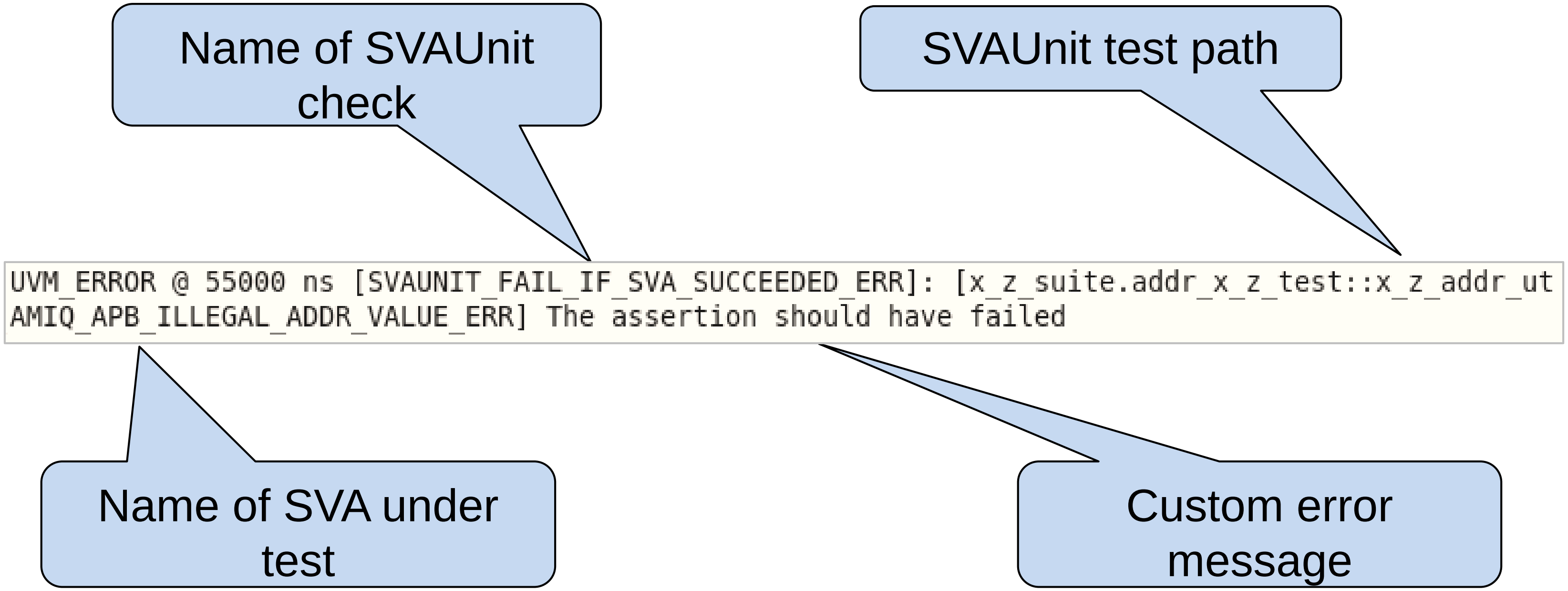
Instantiate test in Test Suite

Register tests in test suite

Simulate

Scan report

Error reporting



Name of SVAUnit check

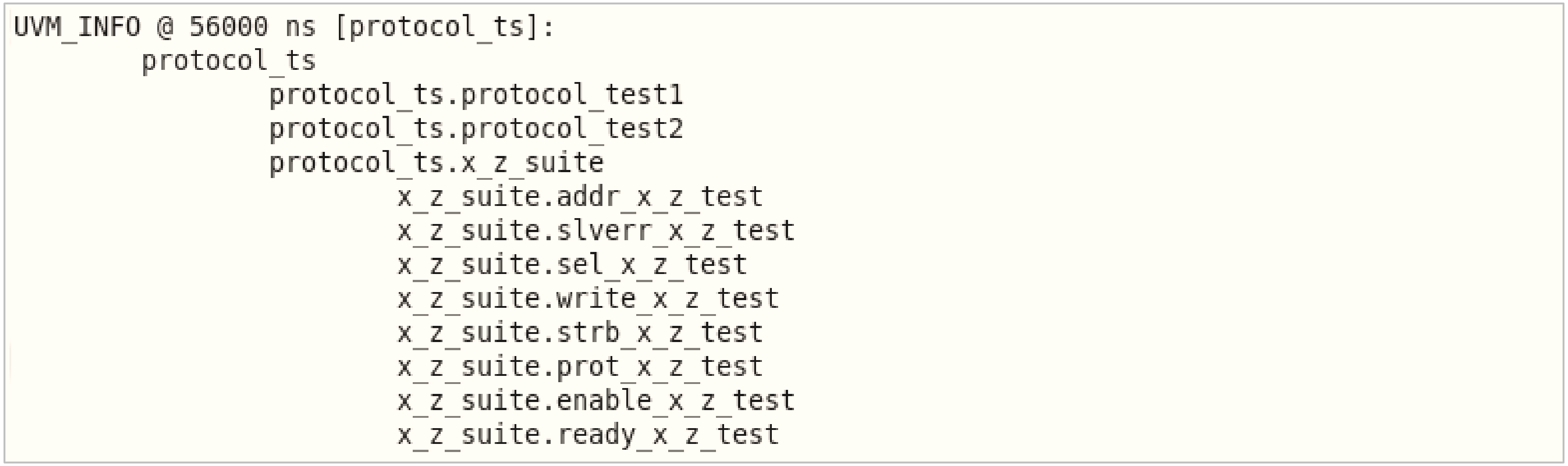
SVAUnit test path

```
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
```

Name of SVA under test

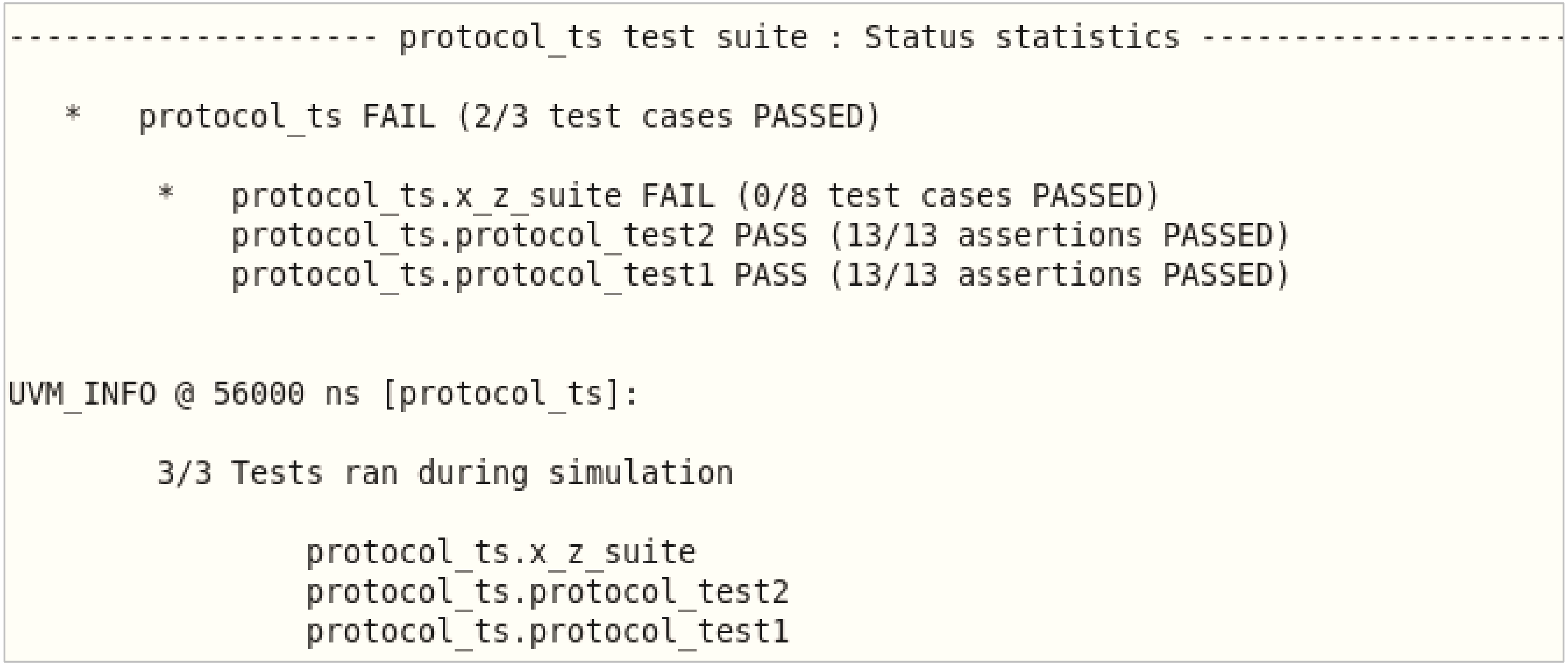
Custom error message

Hierarchy report



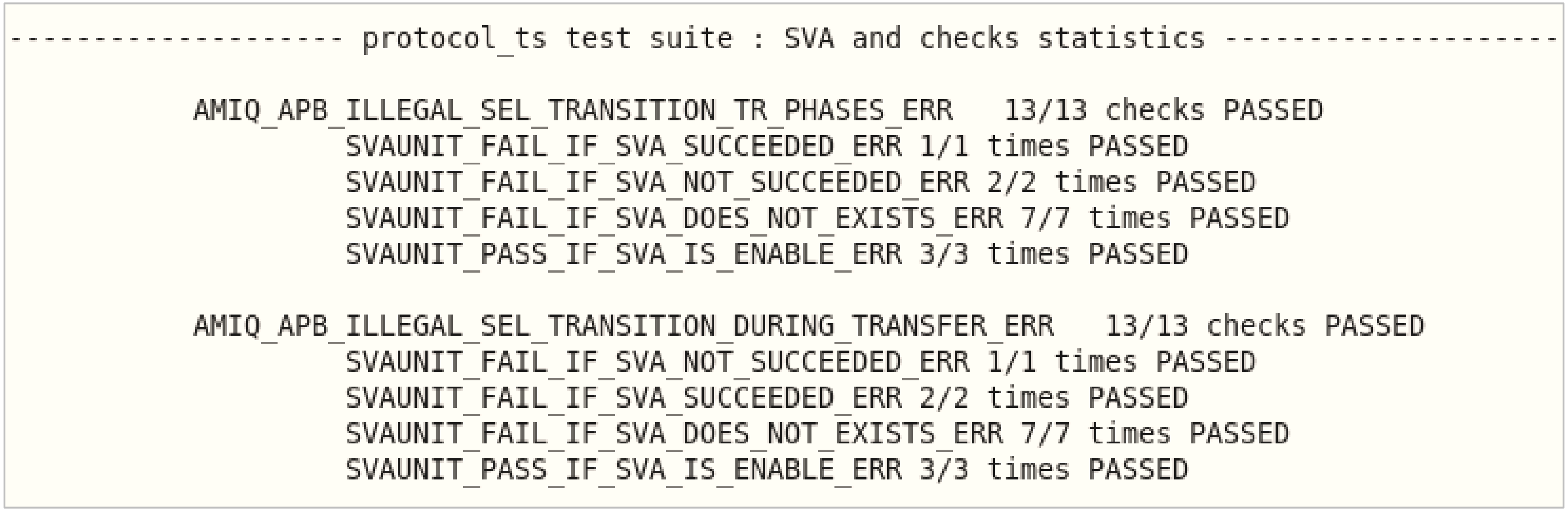
```
UVM_INFO @ 56000 ns [protocol_ts]:
        protocol_ts
                protocol_ts.protocol_test1
                protocol_ts.protocol_test2
                protocol_ts.x_z_suite
                        x_z_suite.addr_x_z_test
                        x_z_suite.slverr_x_z_test
                        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 |=> c

```
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

- `a ##[1:4] b |-> ##[1:3] c`

```
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.)

- a ##[1:4] b |-> ##[1:3] c

```
repeat (test_loop_count) begin
    // Generate invalid delays for asserting b and c signals
    randomize(delay_for_b inside {0, [1:4], [5:10]}, delay_for_c inside {0,[4:10]});
    interface.a  <= 1;

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

    if (delay_for_b >= 5)
      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 inside [1:4])
      vpiw.fail_if_sva_succeeded("MULTITHREAD_ASSERT",
                "The assertion should have failed!");
end
```

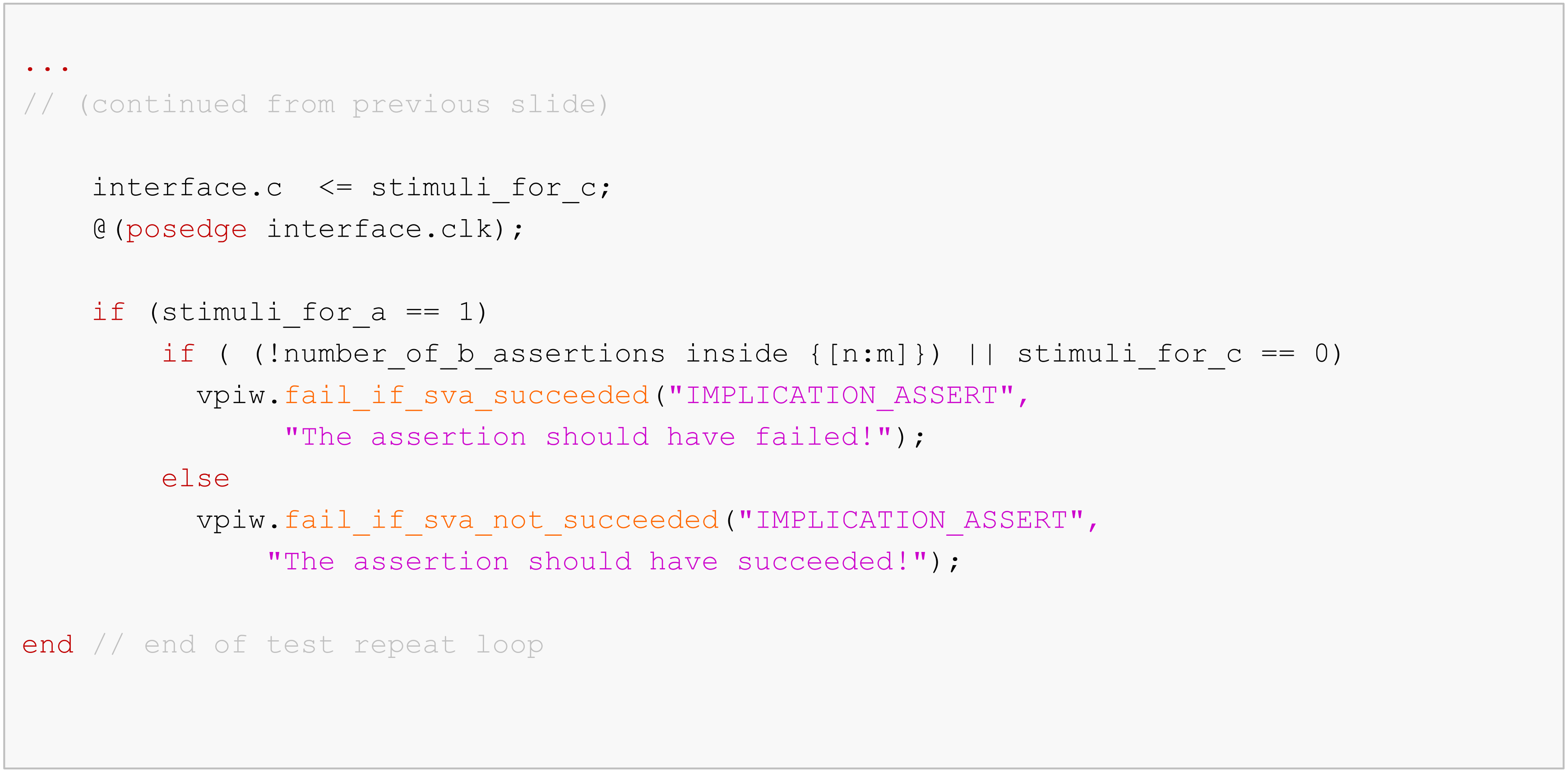
Consecutive repetition

- a |=> b[*n:m] ##1 c

```
repeat (test_loop_count) begin
    randomize(stimuli_for_a, stimuli_for_c, number_of_b_cycles inside {[n:m]});

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

    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 inside {[n:m]})
      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

- `a |=> b[*n:m] ##1 c`



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

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

    if (stimuli_for_a == 1)
        if ( (!number_of_b_assertions inside {[n:m]}) || 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
```

Sequence disjunction

- a |=> (b ##1 c) or (d ##1 e)

```
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 ##1 c)
          SVA state check based on branch stimuli

    end
    begin
              Stimuli for branch: (c ##1 d)
          SVA state check based on branch stimuli

    end
    join
end
```

Sequence disjunction (contd.)

- a |=> (b ##1 c) or (d ##1 e)

```
...

  // 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
```

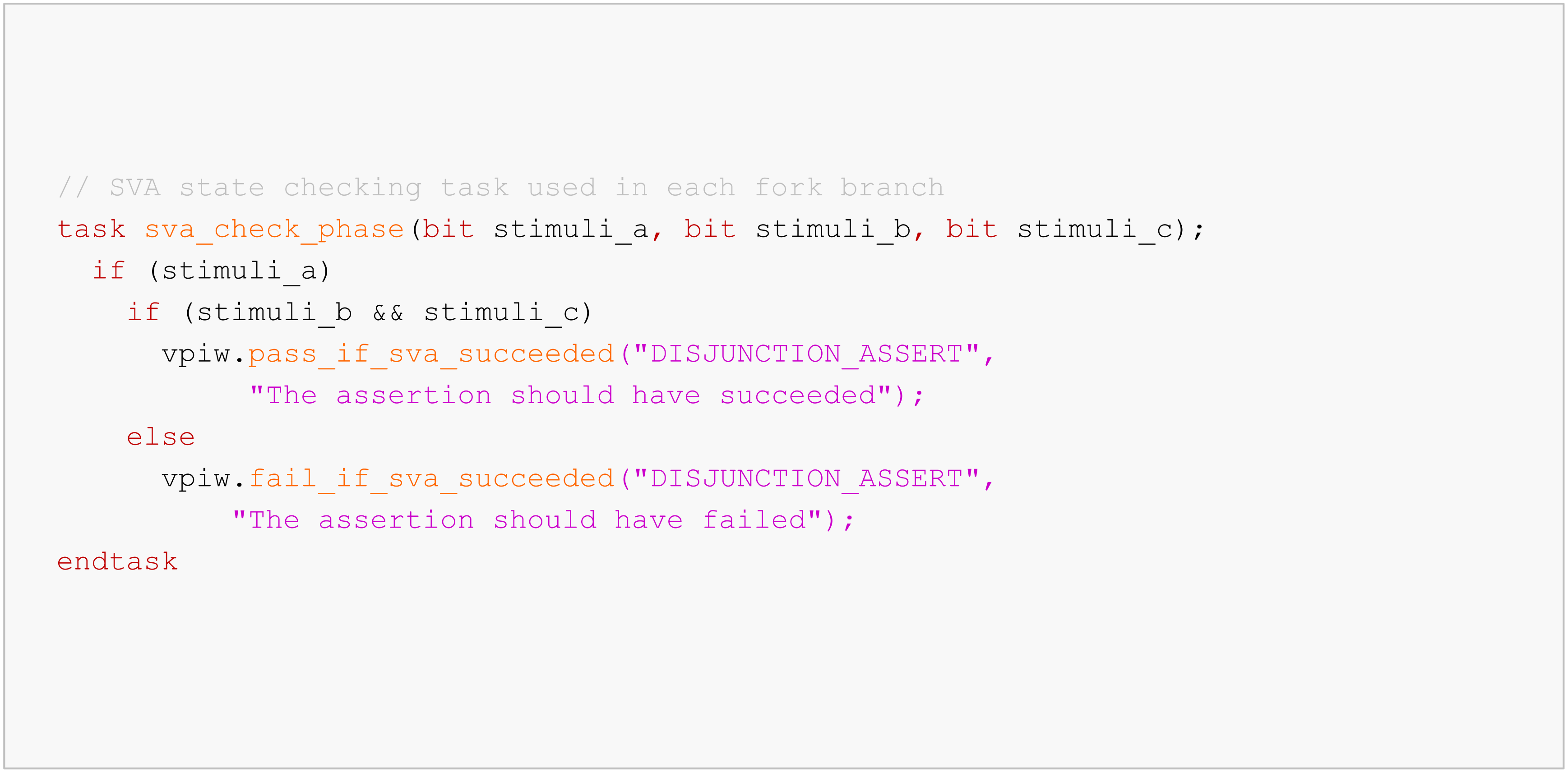
Sequence disjunction (contd.)

- `a |=> (b ##1 c) 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
```

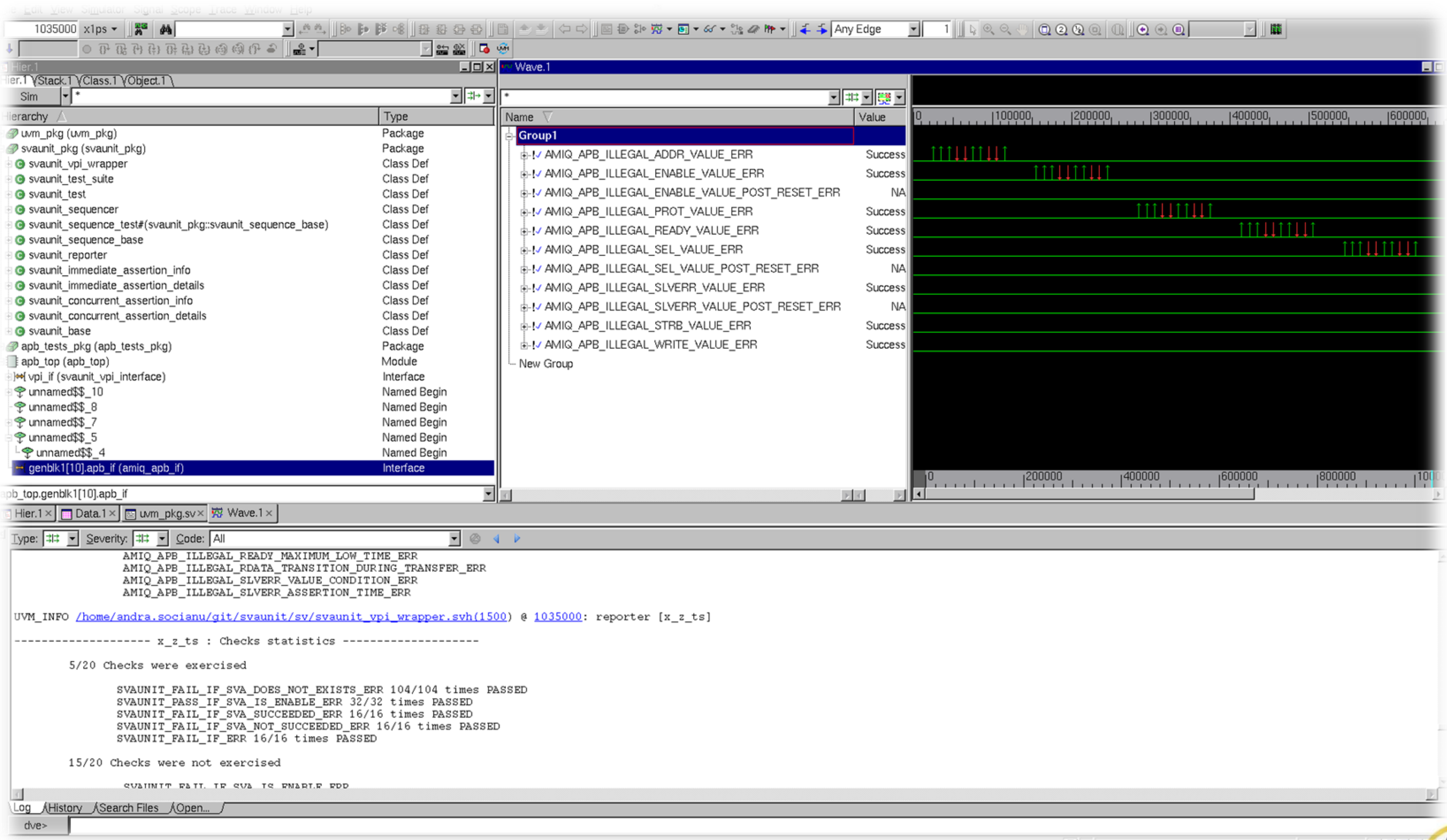
Sequence disjunction (contd.)

- `a |=> (b ##1 c)`



```
// 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

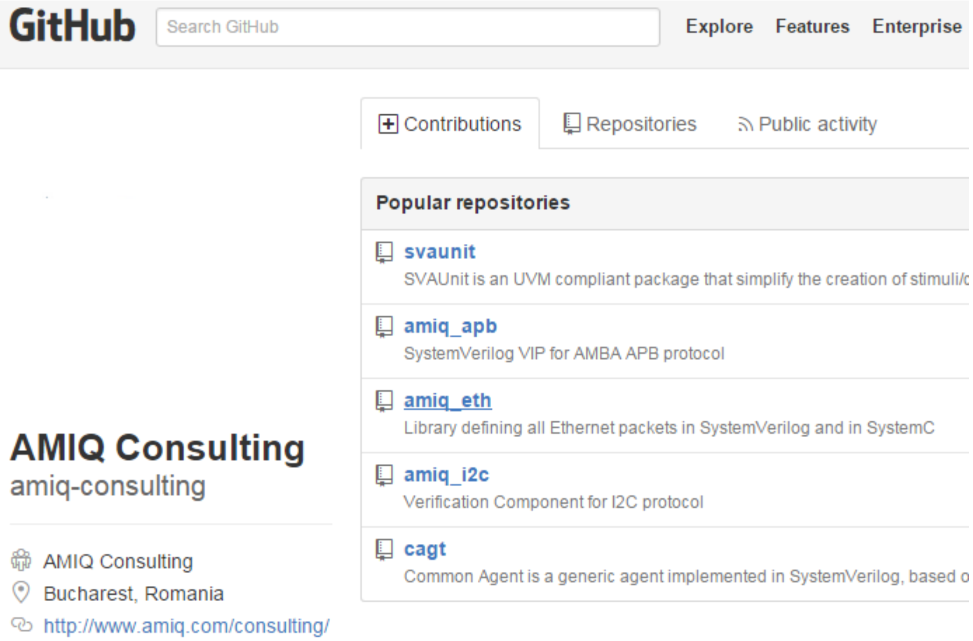


Hierarchy	Type
uvm_pkg (uvm_pkg)	Package
svaunit_pkg (svaunit_pkg)	Package
svaunit_vpi_wrapper	Class Def
svaunit_test_suite	Class Def
svaunit_test	Class Def
svaunit_sequencer	Class Def
svaunit_sequence_test#(svaunit_pkg::svaunit_sequence_base)	Class Def
svaunit_sequence_base	Class Def
svaunit_reporter	Class Def
svaunit_immediate_assertion_info	Class Def
svaunit_immediate_assertion_details	Class Def
svaunit_concurrent_assertion_info	Class Def
svaunit_concurrent_assertion_details	Class Def
svaunit_base	Class Def
apb_tests_pkg (apb_tests_pkg)	Package
apb_top (apb_top)	Module
vpi_if (svaunit_vpi_interface)	Interface
unnamed$$_10	Named Begin
unnamed$$_8	Named Begin
unnamed$$_7	Named Begin
unnamed$$_5	Named Begin
unnamed$$_4	Named Begin
genblk1[10].apb_if (amiq_apb_if)	Interface

Name	Value
Group1	
AMIQ_APB_ILLEGAL_ADDR_VALUE_ERR	Success
AMIQ_APB_ILLEGAL_ENABLE_VALUE_ERR	Success
AMIQ_APB_ILLEGAL_ENABLE_VALUE_POST_RESET_ERR	NA
AMIQ_APB_ILLEGAL_PROT_VALUE_ERR	Success
AMIQ_APB_ILLEGAL_READY_VALUE_ERR	Success
AMIQ_APB_ILLEGAL_SEL_VALUE_ERR	Success
AMIQ_APB_ILLEGAL_SEL_VALUE_POST_RESET_ERR	NA
AMIQ_APB_ILLEGAL_SLVERR_VALUE_ERR	Success
AMIQ_APB_ILLEGAL_SLVERR_VALUE_POST_RESET_ERR	NA
AMIQ_APB_ILLEGAL_STRB_VALUE_ERR	Success
AMIQ_APB_ILLEGAL_WRITE_VALUE_ERR	Success
New Group	

```
                AMIQ_APB_ILLEGAL_READY_MAXIMUM_LOW_TIME_ERR
                AMIQ_APB_ILLEGAL_RDATA_TRANSITION_DURING_TRANSFER_ERR
                AMIQ_APB_ILLEGAL_SLVERR_VALUE_CONDITION_ERR
                AMIQ_APB_ILLEGAL_SLVERR_ASSERTION_TIME_ERR

UVM_INFO /home/andra.socianu/git/svaunit/sv/svaunit_vpi_wrapper.svh(1500) @ 1035000: reporter [x_z_ts]

-------------------- x_z_ts : Checks statistics --------------------

        5/20 Checks were exercised

               SVAUNIT_FAIL_IF_SVA_DOES_NOT_EXISTS_ERR 104/104 times PASSED
               SVAUNIT_PASS_IF_SVA_IS_ENABLE_ERR 32/32 times PASSED
               SVAUNIT_FAIL_IF_SVA_SUCCEEDED_ERR 16/16 times PASSED
               SVAUNIT_FAIL_IF_SVA_NOT_SUCCEEDED_ERR 16/16 times PASSED
               SVAUNIT_FAIL_IF_ERR 16/16 times PASSED

        15/20 Checks were not exercised

                SVAUNIT_FAIL_IF_SVA_IS_ENABLE_ERR
```

Availability



GitHub Search GitHub Explore Features Enterprise

Contributions Repositories Public activity

AMIQ Consulting
amiq-consulting

AMIQ Consulting
Bucharest, Romania
http://www.amiq.com/consulting/

Popular repositories

svaunit
SVAUnit is an UVM compliant package that simplify the creation of stimuli/c

amiq_apb
SystemVerilog VIP for AMBA APB protocol

amiq_eth
Library defining all Ethernet packets in SystemVerilog and in SystemC

amiq_i2c
Verification Component for I2C protocol

cagt
Common Agent is a generic agent implemented in SystemVerilog, based o

- 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

- Unit testing methodology in assertion verification
- Use SVAUnit to decouple 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
- Easy-to-use and flexible API
- Speed up verification closure
- Boost verification quality

Q & A

?