Best Practice of Assertion Based Verification

Flora Gao
LSI Corporation
Agenda

- Overview of Cadence Metric-Driven Verification
- Customized Assertion Planning and Implementation
  - Assertion Planning
    - Immediate assertion in direct test case.
    - Concurrent assertion for black-box verification.
    - Concurrent assertion for white-box verification.
    - Assertion-Based OVM Test Bench
  - Assertion Implementation Tips
    - Implement all planned assertions with empty contents at first
    - Develop assistant signals like sampling clock, disable condition signal, etc.
    - Turn off a bunch of assertions or individual assertions without recompiling
    - Assertion developed for white-box approach
    - Avoid large property or sequence, break them into smaller ones
    - Small tips
- Conclusions
Metric-Driven Verification (MDV)
Planning with unified verification metrics

Testbench simulation, formal, hybrid, HW/SW co-sim, equivalence checking, CDC, constraint checking, LPV, MSV, sim acceleration, emulation

© 2009 Cadence Design Systems, Inc. All rights reserved.
Plan Assertions through Enterprise Planner

- Immediate assertion as checks in direct test case
  - Add description of immediate assertion in planned test case.
  - Easy to develop and check feature directly at early stage.

- Concurrent assertion as checks for black box verification.
  - Check the black box features automatically
    - Implemented in interface.
    - Label the assertion with meaningful name.

- Concurrent assertion as checks for white box verification.
  - Develop assertion that bound to specific design module.
  - Suit for digital IP, easy to reuse in other project.
Plan Immediate Assertion in Direct Test

- Immediate assertion in direct test case
  - Add description of immediate assertion in planned test case.
  - Easy to develop and check feature directly at early stage.

```plaintext
assert (op_FHC_Enable_seq.val == 0) else $error("CheckPoint2 Failed: MSTR_HT doesn't reset to zero when LowSupply fault is detected!");
```
Plan Concurrent Assertion for Black Box Verification.

- Concurrent assertion for black box verification.
  - Check the black box feature automatically
  - Implement in interface.
  - Label the assertion with meaningful name.

```verilog
property p_FHC_PWR_MODE_DEC;
  @(e_fhc_reg_update) disable iff (VS_if.TBCFG_if.ASSERT_OFF_fhc)
  local_FHC_POWER_MODE == VS_if.DT_if.FHC_PowerMode;
endproperty

a_FHC_PWR_MODE_DEC_CHK: assert property (p_FHC_PWR_MODE_DEC) else
  $error("%m, FHC_PowerMode decode error.");
```
Concurrent Assertion For White Box Verification.

- Concurrent assertion for white box verification.
  - Implement in assertion module
  - Bind assertion module with specific design module.
  - Suitable for IP design and could be reused in multiple projects

```
bind "DIGITAL_CORE_PATH.FlyHeightControl FlyHeightControl_a_sva FlyHeightControl_sva (*)
```
Assertion Based OVM Test Bench

Immediate assertion check in test case

Concurrent assertion check in interface

Unified Clock management

Unified Reset management

SVA on/off configuration

Concurrent assertion module that bind to design

Test Case

RGM OVC
SPI OVC

A OVC
B OVC

Reset OVC

Clock OVC

test SVA1
test SVA2

DUV
Develop Empty Assertion At Early Verification Stage

- Implement all planned assertions with empty contents at first
  - Local_EMPTY_SVA_FLAG is used to disable it
- Back-annotate the empty assertions to verification plan to hook-up the plan and implementation
- Remove empty assertions gradually with real ones

```verilog
property p_HEAT_DEC_CHK;
  @(posedge local_CLK,5NS) disable iff (ASSERT_OFF_flag || local_EMPTY_SVA_FLAG)
    3;
endproperty

a_HEAT_DEC_CHK: assert property (p_HEAT_DEC_CHK) else
  $error("%N, Heat Decode value error.");
```
Assistant Signal Generation - unified sampling clocks

- Generate unified sampling clocks to avoid use high frequency one

```verilog
interface pa_clk_if();

logic sig_CLK_1NS;
logic sig_CLK_5NS;
logic sig_CLK_1;
logic sig_CLK_2;
logic sig_CLK_3;
logic sig_CLK_4;
logic sig_CLK_5;
logic sig_CLK_6;
logic sig_CLK_7;
logic sig_CLK_8;

endinterface

fork

  `ovn_do_on_with(clk_seq, p_seqencer.clk_seq[CLK_1NS],
    clk_seq.clock_name = CLK_1NS; //1ns clock
    clk_seq.clock_high = 500; //ps
    clk_seq.clock_low = 500; //ps
    clk_seq.clock_life = 0)); //ps

  `ovn_do_on_with(clk_seq, p_seqencer.clk_seq[CLK_5NS],
    clk_seq.clock_name = CLK_5NS; //5ns clock
    clk_seq.clock_high = 2500; //ps
    clk_seq.clock_low = 2500; //ps
    clk_seq.clock_life = 0)); //ps

  `ovn_do_on_with(clk_seq, p_seqencer.clk_seq[CLK_1],
    clk_seq.clock_name = CLK_1; //10ps clock
    clk_seq.clock_high = 5; //ps
    clk_seq.clock_low = 5; //ps
    clk_seq.clock_life = 0)); //ps

  `ovn_do_on_with(clk_seq, p_seqencer.clk_seq[CLK_5],
    clk_seq.clock_name = CLK_5; //500ps clock
    clk_seq.clock_high = 250; //ps
    clk_seq.clock_low = 250; //ps
    clk_seq.clock_life = 0)); //ps

join none
```

property p_BLK_A_FEATURE_1 CLK;
  //posedge clk_if.sig_CLK_5NS) disable iff (CFG_if.ASSERT_OB_blocks []
  sig_A && sig_B |-> #(0:15) sig_C;
endproperty
Assistant Signal Generation - disable assertion during power on reset

• Generate power on done signal to disable assertion during 1st power on reset

```verilog
logic local_POR_DONE;

initial begin
    local_POR_DONE = 0;
    @(posedge RST_if.reset);
    @(negedge RST_if.reset);
    local_POR_DONE = 1;
end

property p_BLK_A_FEATURE_1_CHK;
    @(posedge CLK_if.sig_CLK_DNS) disable iff (CFG_ifASSERT_OFF_blocks || local_POR_DONE)
    sig_A && sig_B |-> #4(0:15) sig_C;
endproperty
```
Assistant Signal Generation - simplify property or sequence.

- Generate assistant signal to help simplify property or sequence.

```verilog
property p_BLK_A_FEATURE_2_CHK;
@ (posedge CLK_if.sig_CLK_SNS) disable iff (CFG_if ASSERT_OFF blocka ||
| local_POR_DONE)
$rose (sig_A && (sig_B & sig_C | sig_D |!sig_B)) |-> #[0:15] sig_F;
endproperty
```

```verilog
logic local_A_VALID;
assign local_A_VALID = (sig_B & sig_C | sig_D |!sig_B);
property p_BLK_A_FEATURE_2_CHK;
@ (posedge CLK_if.sig_CLK_SNS) disable iff (CFG_if ASSERT_OFF blocka ||
| local_POR DONE)
$rose (sig_A && local_A_VALID) |-> #[0:15] sig_F;
endproperty
```
Turn Off Assertions without Recompiling

- Using "$plusargs" to control a variable used in disable condition.

- Add "$plusargs" as irun command line option to disable assertions without recompile the design

```verilog
module pa_tb_config(
    output reg ASSERT_OFF_all,
    output reg ASSERT_OFF_blocks,
    output reg ASSERT_OFF_blockb
);

    initial begin
        if (0) ASSERT_OFF_all = 1;
        else
            ASSERT_OFF_all = 0;
        if (1) ASSERT_OFF_blocks = 1;
        else
            ASSERT_OFF_blocks = 0;
        if (0) ASSERT_OFF_blockb = 1;
        else
            ASSERT_OFF_blockb = 0;
    end

    ASSERT_OFF_blocks = ASSERT_OFF_all | ASSERT_OFF_blocks;
    ASSERT_OFF_blockb = ASSERT_OFF_all | ASSERT_OFF_blockb;

endmodule

property p_BLK_A_FEATURE_1_CHK; @posedge CLK_if.sig_CLK_5NB disable iff (FIFO_if Assertion OFF blocks || local_PGR_DONE) sig_A << sig_B |-> #[0:15] sig_C;
endproperty
```
 Assertion Implementation

• Assertion developed for white-box approach
  – Suitable for digital IP, easy to reuse in another project.
  – White-box assertion developed by verification engineer should bound to the module.

• Tips for FSM assertion development
  – Check if enter illegal state.
  – Check output code at each state.
  – Transient state duration check.
  – State transition check.
    Forward check: If condition exist, the FSM should transfer to proper next state.
    Backward check: If FSM state transferred, check the proper condition should exist.
Break Large Property or Sequence into smaller ones.

- Simplify property to easy to read
- Smaller sequences can be reused in multiple properties/sequences.
- Large complex property effect simulation performance.
- Easier to debug than a long chain of expressions in a single property

```vhdl
// Bad Example
property p_bus_req_prop;
  @ (posedge clk) req #10 grant #10 !req #1 !grant;
endproperty
a_clk_req_grant: assert property (p_bus_req_prop);

// Good Example
sequence s_start_bus_req;
  req #1 grant;
endsequence

sequence s_end_bus_req;
  !req #1 !grant;
endsequence

property p_bus_req_prop;
  @ (posedge clk) start_bus_req #10 end_bus_req;
endproperty
a_clk_req_grant: assert property (p_bus_req_prop);
```

```vhdl
sequence s_BLR_A_READ_DAC_EQUAL_ZERO_IN_READ_MODE;
  R0; H_DAC == 0;
endsequence

sequence s_BLR_A_WRITE_DAC_EQUAL_ZERO_IN_WRITE_MODE;
  W0; H_DAC == 0;
endsequence

sequence s_BLR_A_TURN_OFF;
  A_OFF == 1;
endsequence

// Bad Example
sequence s_BLR_A_IS_OFF_WHEN_ACTIVE_READ_DAC_EQUAL_ZERO;
  s_BLR_A_READ_DAC_EQUAL ZERO IN_READ_MODE or
  s_BLR_A_WRITE_DAC_EQUAL_ZERO IN_WRITE_MODE
|=> #2[1] s_BLR_A_TURN_OFF;
endsequence

// Good Example
sequence s_BLR_A_IS_OFF_WHEN_ACTIVE_READ_DAC_EQUAL_ZERO;
  s_BLR_A_READ_DAC_EQUAL ZERO IN_READ_MODE
|=> #2[1] s_BLR_A_TURN_OFF;
endsequence

sequence s_BLR_A_IS_OFF_WHEN_ACTIVE_WRITE_DAC_EQUAL_ZERO;
  s_BLR_A_WRITE_DAC_EQUAL ZERO IN_WRITE_MODE
|=> #2[1] s_BLR_A_TURN_OFF;
endsequence
```
Avoid to Use Large Time Ranges

- The following sequence can result in 8000 separate threads and impact simulation performance severely

```verbatim
sequence s_BLK_A_FEATURE_1_61
  [a ##1 b[*1:8000] ##1 c ##1 d];
endsequence

=>

(a ##1 b[*1] ##1 c ##1 d);
(a ##1 b[*2] ##1 c ##1 d);
......
(a ##1 b[*7999] ##1 c ##1 d);
(a ##1 b[*8000] ##1 c ##1 d);
```
Avoid to Use Infinite Time Ranges

• Using infinite time range in the following sequence won't report a failure if acpt is not available after rdy valid and before simulation end

```
sequence s_HANDSHAKE_CHK;
@ (posedge clk) rdy |-> #1:8 acpt #1:1 acpt;
endsequence
```

• Solution
  – According to overall functional timing spec requirement, provide a timeout value as short as possible to replace infinite time range
  – Leave enough drain time before ending simulation
Small Tips

• The sampled value at a transition edge is the previous value.

• $\text{rose}(A)$ is not equal to “ ! A ##1 A ”
  It may also be triggered by some glitch that we don’t care.

• “A |-> ##3 B ” or “ A[∗3] |-> B” which one is expected behavior
Conclusions

• Highlight of Assertion-Based Verification
  – Automatically and constantly checks behavior, especially combined with random stimulus
  – One of the key metrics in Metric-Driven Verification methodology
  – Close to potential source of bugs
    • No need to propagate error to outputs and back-track to source
    – Travel with the design from project to project, ideal for IP

• Lowlight of Assertion-Based Verification
  – Assertion with bad quality consumes a lot of time to debug false failures and fix the assertion statement itself.
Q & A
Thanks!