Sub-cycle Functional Timing Verification using SystemVerilog Assertions

Anders Nordstrom
andersn@verilab.com
Verilab Inc.

Copyright © 2013 Verilab & SNUG
Agenda

- System Under Verification
- Verification Problem
- Verification Approaches
- Assertions in OVM
- Assertions for Functional Timing Verification
- Examples
- Conclusions

Recommendations: Shown throughout the presentation
System Under Verification

• Standard Dual In-line Memory Module (DIMM)
  – Equal track lengths for data
  – Address to every chip

DQ0
DQ31
DQ63
System Under Verification

- Register PLL Dual In-line Memory Module (RDIMM)
  - Different track lengths for data
  - DUT is RPLL like device

\[ c \approx 300,000 \text{km/s} \]
\[ c \approx 3 \text{cm}/100\text{ps} \]
Verification Problem

- Make sure DQ0 and DQ31 are aligned
- Insert delay in DUT to align them
- Problem: CK to DQ is set (tPDM)
Verification Problem cont.

- Must take PVT variations into account
- DLL adds fixed delay
- Compensates for delay variations
Verification Approaches

- Problem: Vary PVT delay – Ensure programmed tPDM is maintained
- Extend OVM Environment
  - Already generating DDR3 transactions
  - Time stamps on transactions
  - Transaction timing, not signal timing
- Verilog Timing Checks
- SystemVerilog Assertions
Verification Approaches cont.

- Verilog Timing Checks
  - $setup(), $hold(), $width()
  - Inside specify block
  - Variable is specparam
  - Error message not customizable

```verilog
module signal_width_check (input wire signal2check);

specifcation
  specparam min_width = 1250;
  $width(posedge signal2check, min_width);
  $width(negedge signal2check, min_width);
endspecification

e ndmodule // signal_width_check
```

**Warning! Timing violation**

$width( negedge signal2check:73511332340 FS,  : 73512580250 FS,  1260 : 1260 PS );
File: /home/andersn/phy_assert.sv, line = 576
Scope: harness.top.core.ddr_phy.phy_assert.c_glitch  Time: 73512580250 FS
Verification Approaches cont.

- SystemVerilog Assertions ✔
  - Most flexible solution
  - Custom error messages
  - Easy to disable checking
  - Familiar to the team
Adding Assertions to an OVM Environment

- Where should assertions be located?
- How do you access signals in other modules?
- How do you exclude some or all assertions?
- How do you use configuration data?
Location of Assertions

• Assertions verify:
  – DUT I/O Relationships
  – Internal Signal Behaviors

• Inside DUT or Verification Environment?

• Protocol assertions often in respective Interface

• Interfaces can bind to interfaces or modules

Recommendation 1: Put assertions inside interfaces in separate files
Cross Module References

- Accessing signals outside module bound to
- Hierarchical connection in bind

```vhdl
bind ddr_phy ddr_assert my_assert (.c_hclk (ckgen_inst.c_hclk), .*);
interface ddr_assert (input c_hclk,
                     input rst_n);
```

- Direct Hierarchical name access

```vhdl`
define OFFSET_REG harness.card_inst.dut_inst.inst_core.inst_phy.reg_inst
interface ddr_assert (input c_hclk,
                      input [1:0] rc10,
                      input [7:0] ckq_off, ckdb_off);
always @(*)
case (rc10[1:0])
  2'0b00: begin   //DDR800
    ckq_off = `OFFSET_REG.c_reg_ckq_off;
    ckdb_off = `OFFSET_REG.c_phy_ckdb_off;
  end
endcase
`
```

Recommendation 2: Connect all signals explicitly in bind statement, if not connected by .*
Excluding Assertions

• “-assert disable” compile option
  ```
  assert($cast(this.my_parent, get_parent() ))
  ```
  is not executed

Recommendation 3: Use assertions to check DUT signals only

• `ifdef around bind

  ```
  OVM_ERROR @ 0.000 ns: reporter [SVA phy_dll_assert] sdll phy_dll_assert sigo high or low period too short. Got 0 Expected x
  ```

Recommendation 4: Use `ifdef around bind and interface
Excluding Assertions

- SystemVerilog assertion control tasks

```verilog
initial begin
  $assertoff(0, `PHY_ASSERT_PATH.inst_phy);
  $display ("Phy assertions disabled for test foo ...");
```

Recommendation 5: Use $assertoff() to turn off assertions for specific testcases
Configuration Information

- DDR3 Interface: 4 speeds, 2 voltages
- Test environment write RC10 register at startup
- Option1: Use register values in assertions

```verbatim
bind phy ddr_assert my_assert(.rc10(reg_inst.rc10), .*);

interface ddr_assert(input [1:0] rc10);
time sim_period;
always @(*)
    case(rc10[1:0])
        2'b00: sim_period = 2500ps; //DDR3-800

```

- Assertion check and DUT in sync
- If DUT programming fails – assertion may not fail
Configuration Information cont.

- Option 2: Use configuration in test environment

```verilog
class timing_cfg extends ovm_transaction;
rand int sim_period;
...
constraint sim_period_cons {
  (speed == DDR_800) -> sim_period == 2500;
  (speed == DDR_1066) -> sim_period == 1876;
  (speed == DDR_1333) -> sim_period == 1501;
  (speed == DDR_1600) -> sim_period == 1250;
}
...
function void post_randomize();
  harness.card_inst.top.inst_core.inst_phy.my_assert.sim_period = this.sim_period;
endfunction : post_randomize
```

- Ensures that assertion checks test intention

**Recommendation 6:** Use environment configs instead of DUT registers
Assertions for Functional Timing Verification

• Why is this a problem?

\[ \text{a\_example: assert property (@(posedge clk) !(GntA && !ReqA))} \]

- No Reference Clock
- Sampled in preponed region
- Jitter on signals
- Un-even Simulator Support
Sampling Point

- Requirement: “mux inputs to DLL are low immediately following mux select change”

![Waveforms showing sigo_0, sigo_1, and mux_sel]

- Immediate assertion:

```verilog
code
always @(mux_sel)
a_mux_sel: assert (sigo_0==0 && sigo_1==0);
```

- ✔️

```verilog
code
always @(mux_sel)
a_mux_sel: assert (sigo_0==0 && sigo_1==0);
```

- ✗
Signal Jitter Impact

- Requirement: A high pulse on DQSEN must be 3 clock cycles long

```
a_dqsen_length: assert property (@(posedge clk) $rose(DQSEN) |=> DQSEN[*2]);
```

- Assertion has clock, but jitter on DQSEN leads to:
  - false pass
  - false failures
Signal Jitter Impact cont.

- E.g. 2.5ns clock, 100ps jitter -> 7.4ns – 7.6ns OK
- Incorrectly fail DQSEN_short
- Incorrectly pass DQSEN_long

- Need to measure length of DQSEN pulse
Signal Jitter Impact cont.

```verilog
parameter dqsen_length = 7500ps;
parameter dqsen_jitter = 100ps;

always @(posedge DQSEN) dqsen_postime = $time;
always @(negedge DQSEN) dqsen_negtime = $time;

sequence dqsen_length_seq;
  @(posedge DQSEN) 1;
endsequence

a_dqsen_length:
  assert property (@(dqsen_length_seq) disable iff(!reset_b)
      (dqsen_postime - dqsen_negtime) >
      (dqsen_length - dqsen_jitter) &&
      (dqsen_postime - dqsen_negtime) <
      (dqsen_length + dqsen_jitter));
```

Recommendation 7: Measure signal transition times instead of using a clock if jitter is modeled.
Incorrect Error Messages

- Requirement: Time for CLK to ODT change < \text{ck2db\_max}

```verilog
always @ (posedge CLK)     db_start_time = $time;
always @ (ODT)             ODT\_delay = $time - db_start_time;

a\_ODT\_delay\_max: assert property (@(ODT\_delay)
                             ODT\_delay <= \text{ck2db\_max})
else ovm\_report\_error("ddr\_assert",
                         $psprintf("ODT\_delay (%t) is longer than \text{ck2db\_max} (%t)",
                                ODT\_delay, \text{ck2db\_max}));
```

- Example of assertion error message:

```
OVM\_ERROR @ 74652.923 ns: reporter [ddr\_assert]
ODT\_delay (1.048 ns) is longer than \text{ck2db\_max} (1.109 ns)
```
Incorrect Error Messages cont.

- Assertion checks preponed value (1250) and fails
- $psprint shows current value (1048) after change

```plaintext
else ovm_report_error("ddr_assert",
    $psprintf("ODT_delay (%t) is longer than ck2db_max (%t)",
        $sampled(ODT_delay), ck2db_max));
```

Recommendation 8: Use $sampled() to display the correct value in error messages, when the value checked is the same as the assertion trigger
Simulator Support

• Requirement: No glitch < \(\frac{3}{8}\) of clock period on DLL output allowed
• Multiple clock property

```verilog
property p_no_glitch (start_event, end_event, min_delay);
time start_time;

@ (start_event) (1, start_time = $time()) =>
@end_event (($time - start_time) > min_delay);
endproperty
```

• Doesn’t work reliably in all simulators
• Combination of multiple clocking events and $time in property sometimes cause false failures
Simulator Support cont.

- Calculate pulse width using $time outside assertion

```verilog
always @(posedge sigo) posedge_time = $time;
always @(negedge sigo) negedge_time = $time;

wire signed [31:0] period_delta = ((negedge_time - posedge_time) < 0) ?
    (posedge_time - negedge_time) :
    (negedge_time - posedge_time);

a_no_glich: assert property (@(period_delta) (period_delta > sdl_width))
    else ovm_report_error("SVA phy_sdl_assert",
        $psprintf("sdl %m sigo high or low period too short. Got %d Expected %d", $sampled(period_delta), sdl_width));
```

**Recommendation 9:** Avoid using $time directly inside assertions
Clock Jitter Impact

- Requirement: Input clock (CK) to output clock (Y0) programmable offset \( t_{\text{STAOFF}} \) should be stable over PVT variations

![Diagram](image)

1. \( t_{\text{staoff}} \) = propagation delay for clock signal (rising CK input clock edge to rising Yn output clock edge).

JEDEC Standard No. xx-x - last edited: December 4, 2008 3:52 pm

Copyright © 2013 Verilab & SNUG
• Jitter on Y0, need to compare to range [ck2y_min : ck2y_max]
  – ck2y_min = ck2y – allowed_jitter
  – ck2y_max = ck2y + allowed_jitter

OVM_ERROR @ 236415.407 ns: reporter [SVA phy_assert] tSTAOFF (0.000 ns)
  is not between ck2y_min (1.793 ns) and ck2y_max (1.913 ns)
Clock Jitter Impact cont.

- Clocks in Phase not considered
  - Fails for tSTAOFF == 0

\[
a_{tSTAOFF}: \text{assert property } (@(tSTAOFF) \text{ disable iff } (reset) \\
(tSTAOFF \geq ck2y_{\text{min}}) \&\& (tSTAOFF \leq ck2y_{\text{max}}))
\]

else ovm_report_error("SVA phy_assert", \\
psprintf("tSTAOFF (\%t) is not between ck2y_{\text{min}} (\%t) \\
and ck2y_{\text{max}} (\%t)", $sampled(tSTAOFF), ck2y_{\text{min}}, ck2y_{\text{max}}));

- Jitter may cause order of edges to move
- Need to consider two cases:
  - Programmed offset greater than jitter (ck2y > jitter)
  - Programmed offset less than jitter (ck2y < jitter)
Clock Jitter Impact cont.

- Case 1: Programmed offset > allowed jitter

- $Y_0$ edge between A and B
Clock Jitter Impact cont.

Example: clock period == 1875ps, ck2y == 1853ps, allowed jitter == 60ps
ck2y_min == 1793ps,  ck2y_max == 1913ps

- tSTAOFF == 1800ps
  - 1800 > 1793 && 1800 < 1913

```
always @ (posedge CK)
    start_time = $time;
@ (posedge Y0);  tSTAOFF = $time – start_time;

a_tSTAOFF_delay1: assert property (@(tSTAOFF) disable iff(reset)
    ((ck2y > jitter) |->
      ((tSTAOFF > ck2y_min) && (tSTAOFF < ck2y_max)) ^
      (((tSTAOFF + ck2y) > ck2y_min) && ((tSTAOFF + ck2y) < ck2y_max))))
else ovm_report_error("SVA phy_assert",
    $psprintf("tSTAOFF (%t) is not between ck2y_min (%t) and ck2y_max (%t)",
        $sampled(tSTAOFF), ck2y_min, ck2y_max));
```
Clock Jitter Impact cont.

- Case 2: Programmed offset < allowed jitter

- Y0 edge between 0 and B or A and clock period
Clock Jitter Impact cont.

- Example: clock period == 1875ps, ck2y == 50ps
  allowed jitter == 60ps
  ck2y_min == -10ps, ck2y_max == 110ps

- tSTAOFF == 70ps
  - 70 >= 0 && 70 < 110

Recommendation 10: Consider all cases of clock phases with jitter – separate assertions
Conclusions

- SVA successfully used to verify DDR3 functional timing
- Found bugs missed in OVM only simulations
- Several issues encountered
  - Integrating with OVM
  - SVA behavior
  - DUT behavior e.g. jitter
- 10 recommendations developed
1: Put assertions inside interfaces in separate files
2: Connect all signals explicitly in bind statement, if not connected by . *
3: Use assertions to check DUT signals only
4: Use `ifdef around bind and interface
5: Use $assertoff() to turn off assertions for specific testcases
6: Use environment configs instead of DUT registers
7: Measure signal transition times instead of using a clock if jitter is modeled.
8: Use $sampled() to display the correct value in error messages, when the value checked is the same as the assertion trigger
9: Avoid using $time directly inside assertions
10: Consider all cases of clock phases with jitter – separate assertions